UNIVERSITÀ DEGLI STUDI DI MILANO

CORSO DI DOTTORATO IN INFORMATICA XXVIII (28^{th}) CICLO

TESI DI DOTTORATO DI RICERCA

A Pyramidal Approach for Designing Deep Neural Network Architectures

INF/01

CANDIDATO
Ihsan Ullah

RELATORE
Prof. Alfredo Petrosino

COORDINATORE DEL DOTTORATO
Prof. Paolo Boldi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
University of Milan

February 2017

Developing an intelligent system, capable of learning discriminative high-level features from high dimensional data lies at the core of solving many computer vision (CV) and machine learning (ML) tasks. Scene or human action recognition from videos is an important topic in CV and ML. Its applications include video surveillance, robotics, human-computer interaction, video retrieval, etc. Several bio inspired hand crafted feature extraction systems have been proposed for processing temporal data. However, recent deep learning techniques have dominated CV and ML by their good performance on large scale datasets.

One of the most widely used deep learning technique is Convolutional neural network (CNN) or its variations, e.g. *ConvNet, 3DCNN, C3D*. CNN kernel scheme reduces the number of parameters with respect to fully connected Neural Networks. Recent deep *CNNs* have more layers and more kernels for each layer with respect to early *CNNs*, and as a consequence, they result in a large number of parameters. In addition, they violate the pyramidal plausible architecture of biological neural network due to the increasing number of filters at each higher layer resulting in difficulty for convergence at training step.

In this dissertation, we address three main questions central to pyramidal structure and deep neural networks: 1) Is it worth to utilize pyramidal architecture for proposing a generalized recognition system? 2) How to enhance pyramidal neural network (*PyraNet*) for recognizing action and dynamic scenes in the videos? 3) What will be the impact of imposing pyramidal structure on a deep CNN?

In the first part of the thesis, we provide a brief review of the work done for action and dynamic scene recognition using traditional computer vision and machine learning approaches. In addition, we give a historical and present overview of pyramidal neural networks and how deep learning emerged. In the second part, we introduce a strictly pyramidal deep architecture for dynamic scene and human action recognition. It is based on the *3DCNN* model and the image

pyramid concept. We introduce a new 3D weighting scheme that presents a simple connection scheme with lower computational and memory costs and results in less number of learnable parameters compared to other neural networks. *3DPyraNet* extracts features from both spatial and temporal dimensions by keeping biological structure, thereby it is capable to capture the motion information encoded in multiple adjacent frames. *3DPyraNet* model is extended with three modifications: 1) changing input image size; 2) changing receptive field and overlap size in correlation layers; and 3) adding a linear classifier at the end to classify the learned features. It results in a discriminative approach for spatio-temporal feature learning in action and dynamic scene recognition. In combination with a linear SVM classifier, our model outperforms state-of-the-art methods in one-vs-all accuracy on three video benchmark datasets (KTH, Weizmann, and Maryland). Whereas, it gives competitive accuracy on a 4^{th} dataset (YUPENN).

In the last part of our thesis, we investigate to what extent *CNN* may take advantage of pyramid structure typical of biological neurons. A generalized statement over convolutional layers from input up-to fully connected layer is introduced that further helps in understanding and designing a successful deep network. It reduces ambiguity, number of parameters, and their size on disk without degrading overall accuracy. It also helps in giving a generalize guideline for modeling a deep architecture by keeping certain ratio of filters in starting layers vs. other deeper layers. Competitive results are achieved compared to similar well-engineered deeper architectures on four benchmark datasets. The same approach is further applied on person re-identification. Less ambiguity in features increase Rank-1 performance and results in better or comparable results to the state-of-the-art deep models.

Keywords: Pyramidal Neural Network, CNN, Action Recognition, Object Classification, Dynamic Scene Recognition.

Parts of this report have been included in the following papers:

Ihsan Ullah and Alfredo Petrosino, "A Strict Pyramidal Deep Neural Network for Action Recognition", in International Conference on Image Analysis and Processing (ICIAP), 2015

Ihsan Ullah and Alfredo Petrosino, "About Convolutional Neural Network with Pyramid Structure", in International Joint Conference on Neural Network (IJCNN), 2016

Sara Iodice, Alfredo Petrosino, and Ihsan Ullah, "Strict Pyramidal Deep Architectures for Person Re-Identification", in Advances in Neural Network 2016

Ihsan Ullah and Alfredo Petrosino, "Spatiotemporal Features Learning with 3DPyraNet", in Advanced Concepts for Intelligent Vision Systems (ACVIS) 2016

Ihsan Ullah and Alfredo Petrosino. "A Deep Pyramidal Neural Network for Spatio-Temporal Feature Learning", To be Submitted in IEEE Transaction on Neural Network and Learning System, 2016.

To my mother,
for her love, prayers and sacrifices.
To my brothers, sisters, brother-in-laws, uncles, and
specially my loving wife and daughter for their continuous prayers,
support and motivation.

Acknowledgments

First and foremost, all praise and thanks to ALLAH, the Almighty for his guidance, protection, sustenance and the innumerable favors which he has bestowed upon me.

For many days I have considered what I might write here, and whom I might thank. After reading the acknowledgments of my friends dissertations over the years, it is clear to me that this page is one of the few true soapboxes allotted to us in life, and it would be tragic, especially for me, to waste it.
I would like to express my special appreciation and thanks to my advisor - Prof. ALFREDO PETROSINO, you have been a tremendous mentor for me by encouraging my research and for allowing me to grow as a research scientist. Your advice on both research and as well as on my professional career have been priceless.

I appreciate the time and effort of thesis external reviewers - Prof. VIRGINIO CANTONI, Prof. SILVIO SAVERESE, and Prof. BRUNO J.T FERNANDES for their valuable feedback and helpful suggestions.

I am highly grateful to University of Milan for awarding me with the scholarship and CVPR Lab, University of Naples 'Parthenope' for providing me an excellent mobility opportunity that had made a positive impact on my life. I am also grateful to my seniors Assist Prof. Alessio Ferone and Assist Prof. Antonio Maratea and junior colleagues in CVPR Lab, who helped me during my research phase

through consistent guidance and motivation. I am thankful to my good friends F. Battistone and S. Iodice for their help and motivation through out my stay at CVPR Lab. Furthermore, I am very grateful to Prof. SILVIO SAVERESE for giving me the opportunity to be a part of the CVGL group at Stanford University.

I am highly thankful to my family members for their kind support and prayers with all of the sacrifices that they have made on my behalf. Last, but not least, I truly appreciate my beloved wife, for your kind love, patience, understanding, consistent encouragement and motivation throughout my life and PhD. Though, miles away, their unconditional support, wishes and prayers have always surrounded me.

Ihsan Ullah

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last two decades, one of the goal of computer vision (CV) research community is to have a smooth representation of the visual world capable of recognizing actions/scenes in videos also in complex scenarios. While the task appears natural and easy for humans, action/scene recognition is very complex for computers and no efficient general solution has been envisaged till now, except for some controlled situations. Indeed, the problem of large-scale realistic action/scene recognition involves many challenging sub-tasks which still remain unsolved.

An action can be measured like a sequence of primitive actions that accomplishes a function or simple purpose, such as jumping, running, walking, or kicking a ball, etc. This task is difficult due to: existence of high variation between the same action done by different people in different styles, the perspectives from which they can be viewed, and possibility of several other external factors, like changing lighting in a scene, color variations for different instances, partial or full occlusions, image resolution due to the distance between the subject and recording camera, and background clutter (Fig. 1.1 and Fig. 1.2). In addition, as shown in Fig. 1.2, there is also a certain degree of intra-class similarity among several action categories.

Similarly, dynamic natural scene classification is an important challenge in the area of automated video understanding. The ability

Figure 1.1: Some real world actions in a video to recognize.

of distinguishing scene is of great interest to video surveillance, robot navigation, video summarization, video segmentation, etc. since it provides hints about the presence of an action, surface, or objects. For example, a forest scene probably contains trees and may include any animal or bird, whereas, a scene from a street mostly contains cars or pedestrians etc. [2]. Similarly, an object, action, their relation, and presence helps to distinguish a specific scene. In a short recorded clip, a set of dynamic patterns and their spatial layout describe a dynamic scene. As an example, a beach scene might be considered by the combination of slowly moving clouds at the top, mid-scene water waves move forward and than backwash, and a foreground of static sandy texture [2, 3, 4]. Fig. 1.3 shows some of the scenes taken from a dynamic scene recognition dataset [5] with camera induced motion. Currently, there is strong interest in spatio-temporal analysis at various levels of complexity, ranging from optical flow and dynamic texture analysis to high-level analysis in terms of scenes of particular events in a video. To deal with these problems, mainly two type of techniques are adopted, i.e. hand-crafted descriptors or learned features in combination with a specific

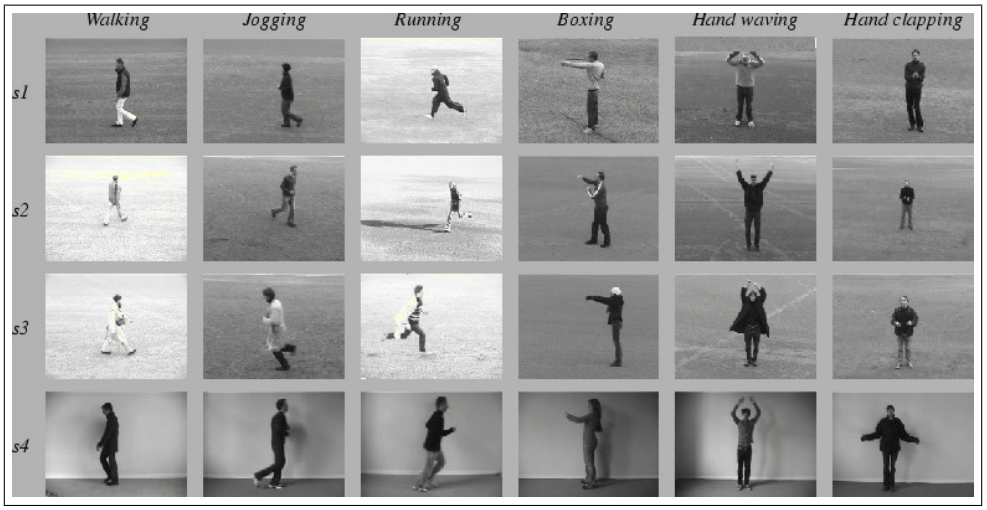


Figure 1.2: Sample Images from KTH Dataset

classifier.

Handcrafted descriptors, such as *HoF* or *HoG* computed on *STIP* [6], proved to be very effective for human action recognition but not for dynamic scenes. Similarly, combination of *HOF+GIST* or *MSOE* [3, 2] or other descriptors shows good result in scene recognition but not for action. Incorporating temporal information usually gets better performance [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19], achieving 90+% accuracy on specific datasets. However, in real-world scenarios represented by large scale datasets [12] with variation in pose, occlusions, illumination changes and their interactions with objects or other subjects in the video, the effectiveness of conventional models significantly degrades.

In the last years, neural network based methods have been developed by going deeper for learning more discriminative and varied features for action recognition [20, 21, 22, 13, 12] and scene recognition [23, 24]. Most of these models are based on the Convolutional neural net (*CNN*) introduced by Y. LeCun et al. [25] in end of 80's.

CNN revealed excellent performance at various tasks in 90's such as hand-written digit classification [25]. Similarly, later it performed well for traffic sign boards recognition [26, 27]. But after a long

Figure 1.3: Samples images from MaryLand dataset. First Row from left to right (Avalanche, Iceberg_Collapse, Waterfall), Second Row from left to right (Fountain, Boiling_Water, Chaotic_Traffic)

struggle, G. Hinton et al. reintroduced *CNN* with recent resurgence of neural networks known as Alex *ConvNet* a deep *CNN* model [12].

The newly proposed deep neural network proved to be an operative key for mining high level features from the input data by winning ImageNet-2012 classification competition. Since than, several works have shown extraordinary performance, and error on ILSVRC-12 is now reduced to 4.7% [28] which is considered better than human accuracy on ILSVRC-12. This performance is brought by several factors; availability of labeled large data i.e. 1.2 million images in ILSVRC-2012 [29], introduction and availability of powerful GPU's and their implementation in parallel programming structure [12], and some new activation functions and regularization techniques that avoids over fitting and local minima problem [12, 28]. More recently, some deep neural architectures are introduced to handle action/scene recognition. As instance, the 3D-convolutional neural network (*3DCNN*) [13] model is introduced for action recognition in surveillance videos; with the help of its 3D kernel not only spatial

but also temporal features are captured.

Anyway, this inspiring progress is still limited in understanding the internal and external operation and behavior of these complex models. How they attain such good performance is still a question mark. Is it the model architecture width and depth, number of parameters, regularization techniques, or availability of data and technology that made this performance possible? Indeed, without a fully understanding of how one can design an architecture, achieving better results on more challenging tasks is reduced to a hit-and-trial concept.

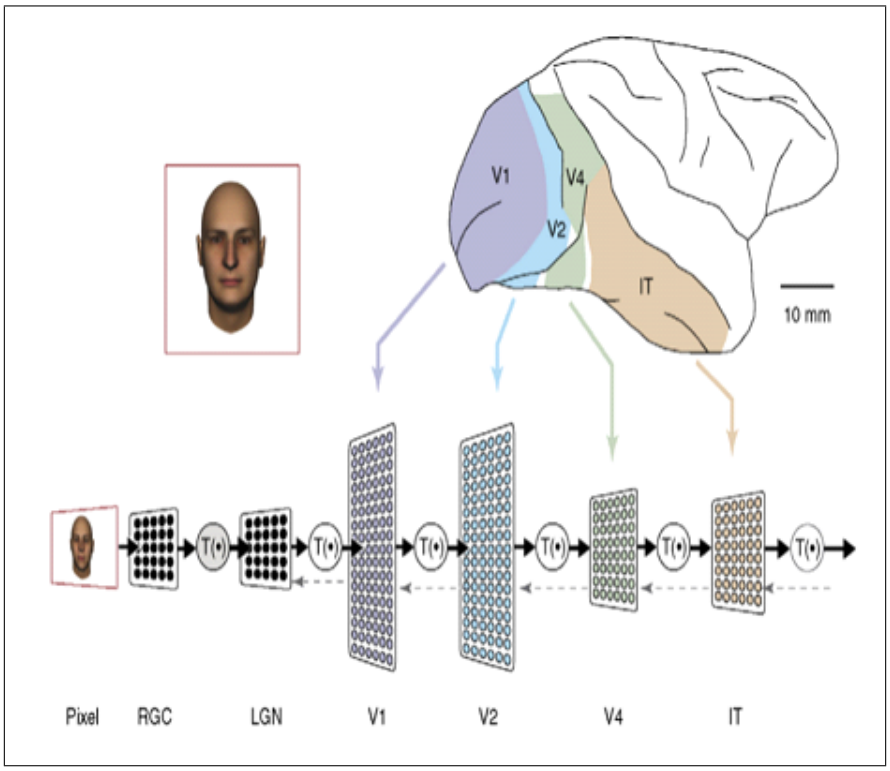


Figure 1.4: Visual Cortex structure of a Human.

A main drawback in deep architectures is that the number of parameters is unarguably a substantial issue in applications. Many researchers are trying to reduce the amount of parameters or solver size on disk, hence, it is infeasible to be implemented in mobile devices. In this regards, in [30] the authors tried to reduce the number

of parameters by avoiding fully connected layers (as majority of parameters are in those layers). Sparsity reduction complex methodologies are used in [31] for refining the trained models, or by learning connections layer wise instead of weights and then retraining the network [30]. On the other hand, these models also violate biological concepts, e.g. rather than decreasing ambiguity, it increases it by increasing the number of maps and parameters.

Literature review on biological studies showed that biological neural network works in pyramid structure [32, 11, 33]. Visual processing is done in stages where each area performs a transformation on its inputs and invariance is built gradually across many successive step as can be seen in Fig. 1.4 [34, 35]. The image passes from retina to lateral geniculate nucleaus (LGN) and reaches the visual cortex divided in different layers. Through these layers, a decision is taken about what the object, person, or action is seen at a specific time [32]. An in depth structure is shown in Fig. 1.5, that shows how a signal path inside a brain of a macaque monkey passes through layers [35]. At each layer, level of abstraction increases as the signals are propagated from the retina to the different regions of the brain. For example, majority of the connections from the LGN go directly to the primary visual cortex. In the primary visual cortex (PVC), simple cortical cells are the first neurons to receive signals which detects lines at different angles from horizontal to vertical direction, whereas complex cortical cells detect motion. At the end, hyper-complex pyramidal cells detect specific bars with specific length and motion. All connections are further transferred to more complex region of brain to detect complex patterns. Literature review demonstrated that some of these neurons are particular in detection of basic shapes like triangles, squares or circles, while other neurons (grandmother cells) are activated when visualizing faces or complex objects.

Pyramids at their simplest are like stack of filtered images with exponentially reduction in their dimensions. Pyramidal structure is natural, compact, and an efficient technique for refinement, learning,

and adaptation of high level features in an algorithm. Several models have been proposed based on the concept of pyramids, e.g. Neocognitron, early LENET, Pyramidal neural network, spatial pyramids, etc. [33, 25, 14, 36]. The term pyramid in deep learning is being used several times. However in all previous works, it is used in a very limited scenario, e.g. either in one layer like the Spatial Pyramid Matching (*SPM*) [36] for Pool layer or in last *Conv* layer. Recently, the model in [37] adopts pyramid structure in its last Conv layer of CNN for face recognition on LFW dataset, achieving 97.5% accuracy. P. Wang et al [38] applied temporal pyramid pooling to enhance and use the temporal structure of videos just like spatial pyramids in [68] where the model incorporated weak spatial information of local features. This pyramidal temporal pooling method showed better results than state-of-the-art two-stream model [39] on HMDB1 dataset. The aim of the thesis is to use, analyze, understand, and emphasize the impact of pyramidal structures in deep learning for actions, dynamic scenes, digits and objects recognition.

Contribution:
In this thesis, mainly a deep learning architecture is proposed to handle both spatial and temporal information at the same time. It is based on a new 3D kernel method and strictly following biological plausible pyramidal structure. The inspiration to propose a 3D structure for strictly pyramidal neural network is taken from two models, i.e. *3DCNN* and *PyraNet*. We adopted dominant characteristics of both the models to examine the power of a biological structure in *DL* approaches.

We divide our thesis in three parts. The first part is proposing a new strictly pyramidal model *3DPyraNet* with new weighting scheme for action/dynamic scene recognition from videos. Secondly, a spatio-temporal feature learning approach using *3DPyraNet* in combination with a linear classifier is proposed. The third module applies pyramidal structure to deep CNN architectures for better understanding of its behavior. Some of the well-known datasets for action and dynamic scene recognition are used to assess power of our proposed models, weighting scheme, and the pyramidal architecture.

More in detail, the parts are organized as follows:

Part I:

- in chapter 2, we describe early neural networks since perceptron till Alex ConvNet, and specifically their connection scheme, network structure, activation functions, weight update rules, supervised training, training protocols, error measures, weight initialization, regularization, early stopping , dropout, weight sharing etc. This includes models like *PyraNet*, *IPyraNet*, *CNN*, *ConvNet*, *GoogleNet*, *3DCNN*, etc.

Part II:

- in chapter 3, we combined deep architecture *3DCNN* and *PyraNet* approach to design the proposed model *3DpyraNet*. Its motivation, background, weighting scheme, structure, and architecture are discussed.
- in chapter 4, we proposed an enhanced version of *3DPyraNet*, named *3DPyraNet-F*, capable to learn spatio-temporal features. Its motivation, background, and architecture are discussed. We show how with the *3DPyraNet-F* learned features, a simple linear classifier can outperform other state-of-the-art models.

Part III:

- in chapter 5, we analyze the impact of imposing pyramidal structure on *CNN* while retaining actual performance as well as reducing the parameter number.
- in chapter 6, we summarize our contributions and give advice about possible future research directions.

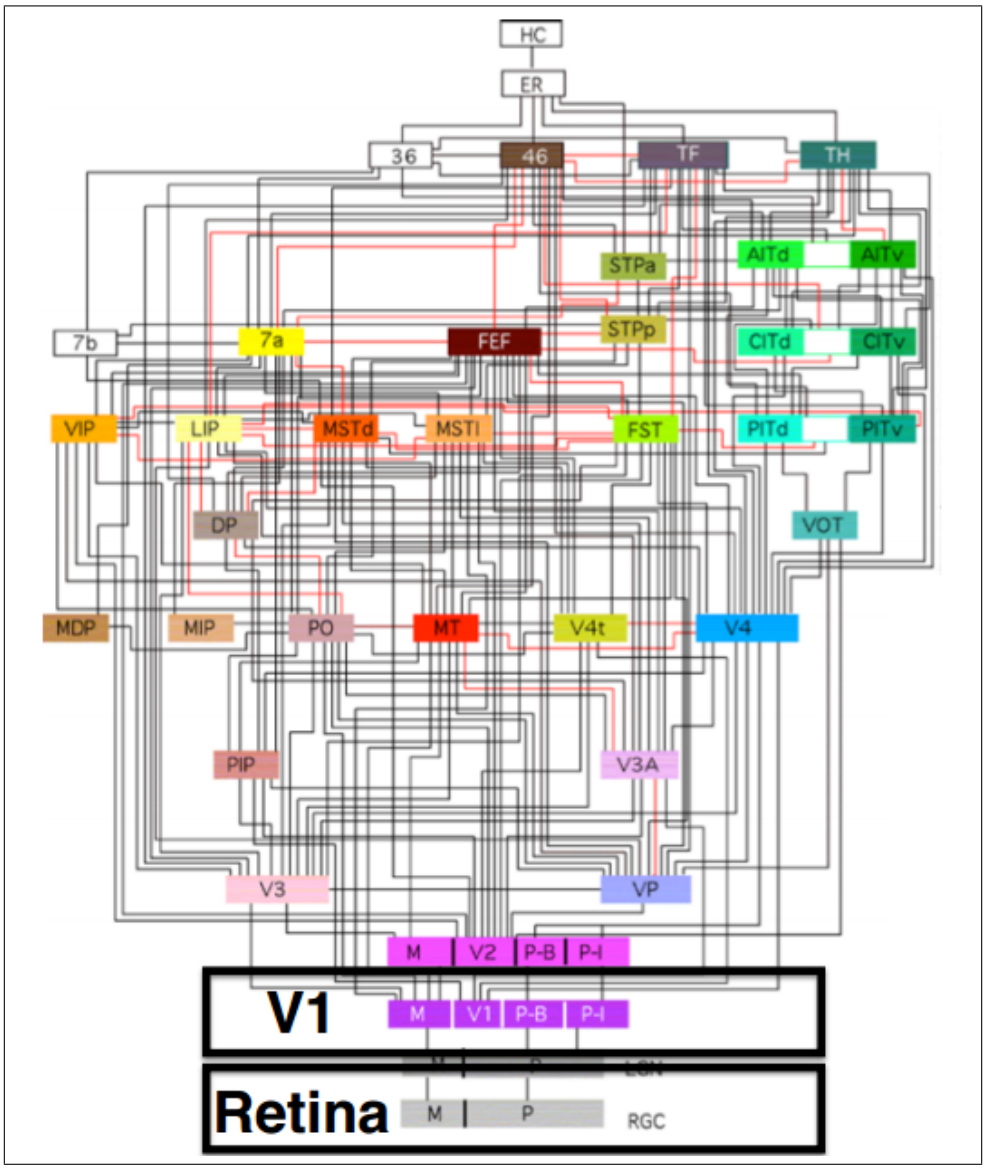


Figure 1.5: Macqake Monkey Brain Deep structure for decision making.

Chapter 2

Related Work

This chapter is divided in four main sections. In the first part, we describe an insight of action/scene recognition. A short review is given on the state-of-the-art work being done for enhancing *AR* and *DSR* from videos in the traditional *CV* and *DL*. The second portion first discusses early *ML* architectures being used since perceptron. It is followed by a detailed background related to pyramidal neural networks (*PNN*) and its key aspects because they are our main focus in this work. Pyramidal structure is used for optimizing the code and enhancing the performance. In the third section of this chapter, we discuss state-of-the-art *CNN* along with few recent very deep flavors of *CNN* for object recognition and for learning spatiotemporal features for (*AR/DSR*), e.g. *AlexNet*, *3DCNN*, *C3D* network etc. Finally, the last part compares and visualize *CNN* and *PyraNet*.

The description helps in understanding a *CNN* architecture for its better modeling and designing of a new architecture. It also helps in proposing a new architecture by studying and keeping in mind the limitations of *CNN* and the advantages of *PyraNet*. One of the main concept we learn from *PyraNet* is adopting its pure pyramidal structure in deep networks. 'Strict/Pure' mean reducing/refining feature maps and their size as the network goes deeper. The *PyraNet* is explained in detail in Sec. 2.2.5 because we are extending it to 3D (i.e. *3DPyraNet*) and imposing its idea on *CNN* to propose strictly

pyramidal *CNN* (i.e. *Spyr_CNN*) models.

2.1 Action and Dynamic Scene Recognition

In the following sub-sections, we give a short overview of hand-crafted and *DL* approaches being proposed for *AR* and *DSR*.

2.1.1 Action Recognition (AR)

AR is one of the challenging research area of *CV* and *ML*. Two types of approaches are adopted to tackle this problem, i.e. Hand-crafted feature extraction approach and features learning approach. In the next subsection, we briefly discuss some of the state-of-the-art hand-crafted feature descriptor/extraction approaches for *AR*. Few of the state-of-the-art feature learning approaches are discussed later in Sec. 2.1.1.2.

2.1.1.1 Hand-Crafted Features for AR

Researchers trust on local hand-crafted features due to their performance and real time processing speed for *AR*. They combine their spatial and temporal information for optimal performance. From several years, low level features are heavily used with much success in *AR*, e.g. spatio-temporal interest points (*STIP*) model that combines temporal Gabor filter and a spatial Gaussian filter in order to extract spatiotemporal key points. This technique is further combined with different descriptors to propose other model such as Harris3D by Laptev et. al. [40]. Their results are still comparable to many state-of-the-art techniques in some databases. Similarly, Cuboid by Dollar et al, 3D-Hessian, Dense sampling, Spatio-temoral regularity based features (*STRF*) with combination of *HOG/HOF*, *HOG3D, SURF*, Extended *SURF*, and *MoSIFT* as feature descriptors show good result for *AR*.

Spatiotemporal features are most favorable when foreground segmentation and tracking are not possible in pre-processing step due to its complexity and large number of frames/videos, e.g. UCF101 dataset [41]. However, according to work done in [42], it is time consuming and not favorable for real-time applications as it takes about 0.9 FPS to 4.6 FPS. A key drawback of local feature based techniques is that the sparse representation such as bag-of-visual-words ($BoVW$) abandons geometric associations of the features and therefore they are less discriminative. In addition, hard-assignment quantization during the codebook creation for $BoVW$, and later clustered by K-means algorithm, also makes the sparse features less discriminative.

Chindler et al. [43] propose a system for human action recognition of a single activity in a video sequence such as walking or jumping. It uses the form features and motion features for feature extraction. A SVM classifier is used to classify both datasets. The main objective of this work is to identify that a short snippets having only 1-7 frames are enough to recognize basic action in a video. The model shows good performance on *WEIZMANN* and *KTH* datasets. Ullah et al. [44] propose a method of classifying human action using the standard Bag of features *(BoF)* in realistic videos. They use segments of videos rather than whole video to extract region level local features using existing techniques and classifying them using a multi-channel SVM. The Hollywood-2 action dataset is used for experiments. The results significantly improved as compare to state of the art techniques. Song et al. [45] focus to elucidate the intra-class variations of human actions and multiple heterogeneous feature representation of videos. Their localized multiple kernel learning (L_MKL) method use to resolve the issues in realistic human AR. The experiments are evaluated on Hollywood-2 and YouTube datasets. The L_MKL achieves the best result and outperformed existing approaches.

Recently, Song et al. [46] present an approach that they termed as the trajectory of surface patch ($ToSP$) to relate the variations of

surface patches on the human body over time. The variety of local patches are clutched using *ToSP* pipeline. The Body Surface Context (*BSC*) uses as a feature extractor forms *ToSP* segments. The NJUST RGB-D [47] and MSRDailyActivity3D [48] public datasets are used for experiments. The *kNN* and *SVM* classifies the testing set and achieves satisfactory results as compared to existing approaches like depth features and trajectory based features.

Zhang et al. [49] evaluate spatial-temporal pyramid sparse coding *(STPSC)* approach for human *AR*. The feature trajectory is used for feature extraction and sparse coding is utilized to calculate paradigm dictionary in-order to attain the lower reconstruction errors. The *KTH* and Hollywood human activities datasets are used for experimentation. The non-linear *SVM* classifier is used for evaluation of dataset that achieves higher result compare to state-of-the-art techniques.

A human action can be observed as combination of consecutive silhouettes over time, where each silhouette registers a pose of this action at a specific instant. Davis et al. [50] reveale after their experiments that a human action can be correctly classified even when it is projected onto a single frame by incorporating partial time element. In [51], human actions are treated via silhouettes as three-dimensional shapes and the Poisson equation properties are adopted to obtain space-time key features. Holistic body shaped and local temporal motions in a silhouettes are combined to encode human actions using a quantized dictionary from space-time windows [52]. L. Shao and X. Chen [53] propose a model in which body poses are sampled from silhouettes and fed into a bag-of-features models. Qu et al. [54] consider the changes between frames and utilize them as transitional features. Sun et al. [55] combine the local 3D-SIFT descriptors and global Zernike motion energy image features by integrating the local and global features.

Since last two decades, many researchers are working on *AR* from videos. They are successful to achieve even more than 90% of accu-

racy. All of these models work in limited scenarios, e.g. one model work only on one dataset or recognition scenario and not on others. However, research community wants some powerful automatic systems that may learn features from input to output by itself. In addition, it may produce generalization, increase accuracy, and should be fast enough to be deployed in real world scenarios due to huge emerging industries. One of the solution for this problem is adopting a *DL* approach for *AR*.

2.1.1.2 Deep Learned approach for AR

The key focus of this work is *NN* based features which is explicitly discussed in the remaining part of this section. Nowadays, as a standard biologically inspired technique, *DL* is one *ML* algorithm that tries to learn high-level perception of data using hierarchical structures. In comparison to traditional hand-crafted features, *DL* achieves more intellectual learning and contains hierarchical feature extraction layers that comprise much more trainable weight parameters than shallow architectures, e.g. kernel machines [56]. Previous models operate on top of hand-crafted action features, e.g. [57, 20], whereas most recent methods proposed end-to-end *AR* frameworks from the pixel-level to classifying them in respective action category.

The foremost natural step to learn action from videos is to consider each frame as an input to a system. Where a supervised *DL* algorithm is applied on each frame to extract action features at the frame-level, e.g. applying a 2D deep *CNN* on individual frame [58, 24]. However, this technique does not result in optimal performance. Therefore, some works tried to learn action from the transformation of frames in an unsupervised manner at first step and later at the final steps they learn the action features in a supervised manner. In this regards, Taylor et al. [20] proposed a deep multi-stage model based on convolutional gated Restricted Boltzmann Machine (*ConvGRBM*). Where, *ConvGRBM* uses 2D convolution to extracts motion-sensitive features from every neighboring frame pairs in an

unsupervised manner. Whereas, the intermediate layer captures the spatio-temporal cues by 3D convolution to learn spatio-temporal filters, followed by a normalization layer, an average spatial pooling layer and an additional max pooling layer that perform pooling in the temporal dimension. Finally, the action representations can be obtained by the fully-connected layers. These upper layers are trained using traditional back-propagation algorithm. B. Chen et al. [21] propose a similar but generative unsupervised model known as space-time deep belief networks (*ST-DBN*). The key idea of this model is to use alternating layers of spatial and temporal convolutional RBM (*CRBM*) to extract long range dependencies from spatial and temporal domain. It also uses weight sharing across all *CRBM* in a layer and measures invariance at each layer for various transformations of the input. The model takes a clip as input and shows superior performance for *AR* in comparison to frame based models. Also, Le et al. [22] use a clip as input to learn features using their unsupervised model. However, they convert the frames using independent subspace algorithm [59] into a feature vector. The network is built by copying the learned network and pasting it to different parts of the input data. Outputs are than treated as the inputs to a new *ISA* network. *PCA* is used to reduce the dimensionality of the feature vector. The reduced set of features from both layers are combined prior to classification.

M. Baccouche et al. [60] propose a model that uses *CNN* and *RNN*. At first step, *CNN* is converted to 3D form that learn spatio-temporal features from raw gray input frames. Once the network converge, feature vector from all clips are given to a newer version (long short term memory (*LSTM*) network) of *RNN* to classify the video in respected category. Similarly, Ji et al. [61, 13] suggest and empirically shows that in practice it is important to provide the network with supplementary information (e.g., optical flow) to facilitate training. Therefore, their propose model takes hard-coded features (in their case gray image, gradient along x-axis and y-axis,

and optical flow along x-axis and y-axis). This 3D model outperformed 2D frame based counterparts with a huge margin. It uses 3D kernels/filters that are extended along the temporal axis to learn spatio-temporal features encoded in adjacent frames. Other than convolution in 3D convolutional layer, pooling and weight-sharing concept also helps in achieving robustness across scale and spatial variations. The model works in a supervised manner that use the back-propagation algorithm to update its network parameters. By analyzing the learned kernels of a *CNN* model shows that the initial layers learn low level features (e.g. edges, or result of Gabor-like filters) whereas higher layers learn high level features (e.g. body parts or high level semantics [62]).

These models have given a baseline for *AR* using *DL*. More recent work use the more deeper variants of AlexNet [12] model to propose their new deep networks. A. Karpathy et al. [24] proposed 4 variants of AlexNet (but small in spatial input size) by fusing the information over temporal dimension through the network in different ways. These variants include single-frame model that consider each frame in the video as an input to classify the action. Late-fusion model include two of single-frame networks each getting input frame that is 13 frames a part from each other while fusing at first fully connected layer. Early-fusion model is an extension of *3DCNN* model, however, the temporal part is more than 3 rather 10. This primary and straight connectivity to pixel data permits the network to specifically detect local motion direction and speed. Whereas, slow-fusion model is combination of late and early fusion. It fuses temporal information gradually through out the network such that higher layer get access to gradually more global information in both spatial and temporal dimensions. This behaves same as the *3DCNN* model but 10 frames as input clip, temporal part of 4 and stride of 2 results in 4 feature maps at higher layer. The model is trained in a month over 1 million youtube videos. This slow-fusion shows the best performance among the four variants.

K. Simonyan and A. Zisserman propose a two-stream AlexNet model [63] with different type of input spatio-temporal input data, i.e. single frame to one stream and optical flow information to the second stream. These two stream are fused after softmax layer by calculating their class score. Based on their experiments, use of optical flow shows significant improvement over raw input frames despite small training data. J. Donahue et al. propose a model [64] similar to model in [60]. This long term recurrent convolutional (*LTRC*) network uses a *CNN* learned feature vector as input to a *LSTM*. However, this model is different in two ways, i.e. i) it integrate 2D CNNs rather than 3D that can be pre-trained on large datasets, ii) *CNN* and *LSTM* are combined into a single model to enable end-to-end fine-tuning. An additional difference can be the variable length input frames to the network. This model shows promising results for action/activity recognition. J. Yue-Hei Ng et al. [65] propose a similar model, however they follow model in [63] as the base structure to extract *CNN* feature vector. This feature vector is given to a feature aggregation module that perform different type of pooling or apply *LSTM* to predict the class predictions for each stream. In the end, fusion of class scores is being done. This model uses two *CNN* architectures, i.e. AlexNet [12] and GoogleNet [66].

D. Tran et al. [23] propose a spatio-temporal feature learning approach that uses *3D-ConvNets* with same number of layers and network structure as AlexNet. The model outperforms all others with small convolution kernels of size $3 \times 3 \times 3$. The model is trained with Sports 1million dataset. The learned model is used as a feature extractor. The extracted features are called *C3D* features that outperformed all other methods on 4 different video analysis tasks with a simple linear *SVM*. However, this model does not show its multi-class classification performance.

2.1.2 Dynamic Scene Recognition (DSR)

A 'scene' represent a place where an action or event take place. However, in comparison to scene recognition from still images where each class label is based only on the spatial properties, dynamic scene classification/recognition/understanding tries to categorize videos into different classes whose semantic labels are derived from the events happening in the scene. *DSR* is an important task in the area of automated image understanding. The capability of a system to distinguish dynamic scenes is very beneficial, as it can help in providing priors for the presence of an action and objects, as well as spatial location and size [2, 67]. A serious challenge for dynamic scene understanding results due to the wide range of naturally occurring phenomena. These phenomenas must be encompassed.

Scene understanding has widely been researched from still images due to their impact on action recognition [68]. Several datasets have been collected to learn indoor and outdoor scenes, e.g. SUN Database [69], Places Dataset [70], Indoor Scene Dataset [71], etc. It involves classifying an image into one of the several given classes. Several *CV* techniques focus on finding appropriate spatial feature descriptors for a given image. However, as previously mentioned, *DSR*/*DSC*/*DSU* tries to categorize videos into different classes based on the semantic labels derived from the events happening in the scene with the passage of time, e.g. the dynamic scenes like 'avalanche' is given its class label based on the breaking, falling down, and movement of ice, and not just based on the spatial attributes of the scene [2, 67]. Therefore, temporal part plays an important role in this regards. In the coming subsections, we explain briefly some of the main traditional hand-crafted and deep learning approaches for *DSR*. In the text we may use *DSC*, *DSR*, or *DSU* interchangeably.

2.1.2.1 Hand-Crafted Features for DSR

This section describes few of the main contributions for improving *DSR* by using hand-crafted local and global features. M. Marszalek et al. [68] start exploiting *DSR* for improving human *AR* in videos. Their model for scenes and actions is based on the bag-of-features frame-work in combination with *SVM*-based classifier. They use 2D and 3D-Harris detector as well as *HoF*, *HoG*, and *SIFT* descriptors for extraction of features. This work highlighted the importance of *DSU* and became the base.

N. Shroff et al. [5] present one of the most challenging dataset known as 'MaryLand-in-the-Wild'. They propose this dataset based on the idea that dynamic attributes can be augmented with spatial attributes of a scene for semantically meaningful classification of a dynamic scene. In addition, one of the main difficulty is the existence of camera induced motion in the videos. They propose Chaos features where each feature is the resultant of certain other features. It is based on the Chaos theory which states that every subsequent point of a given measurement is the result of an entangled combination of influences from all other system variables [72]. Their combination of static and dynamic Chaos features with linear *SVM* outperformed *GIST*, Bag-of-words, Mean *GIST*, and dynamic Chaos.

K. G. Derpanis et al. [2] collect a new dataset called 'YUPENN' dataset. They experiment to find the impact of multiscale orientation measurements on scene classification by systematically evaluating spatial appearance, temporal dynamics and joint spatial appearance and dynamics. Their spatiotemporal oriented energy features (*SOE*) based on the Gaussian filters in combination to nearest neighbor (NN) classifier shows good result for spatial, temporal and spatio-temporal features for their stabilized 'YUPENN' dataset. However, in case of spatio-temporal features of MaryLand-in-the-Wild dataset, their features have slightly bad result in comparison to *Chaos+GIST*

and *HOF+GIST*.

C. Feichtenhofer et al. [18] propose a spacetime descriptor that exploits the corresponding nature of spatial and temporal information, as inspired by prior work on the role of directional features in scene classification. It uses a random forest classifier to learn multi-class categorization of the features. In addition, each video is processed in temporal slices with scale matched favorably to scene dynamics over camera motion. This slicing helps in temporal alignment to be handled as latent information in the classifier and for proficient incremental processing. This model outperforms *Chaos+GIST* and *HOF+GIST* as well as *SOE* on both 'YUPENN' and 'MaryLand' datasets.

In continuation of their work on *DSR*, C. Feichtenhofer et al. [73] propose a model that extracts spatiotemporal oriented primitive features from a temporal subset of the input video. It uses 3D Gaussian third derivative filters as feature descriptor. Than features are encoded into a mid-level representation learned for the task and also steered to extract dynamic pooling energies. It uses a feature vector (FV), improve feature vector (IFV), local linear coding (LLC), and vector quantization (VC) technique for coding. In the end, the encoded features are pooled via a novel dynamic spacetime pyramid that adapts to the temporal image structure, as guided by the pooling energies. The codebook entries for VQ and LLC are quantized and trained using K-means algorithm. Whereas, in case of FV coding, a Gaussian mixture model (GMM) is used. The pooled encodings are concatenated into vectors that work as the final feature vector for online recognition classified by a linear SVM classifier. The model with IFV shows 10% better result than the previous models (*HOF+GIST*, *Chaos+GIST*, *SOE*, *SFA*, *CSO*) on both YUPENN and MaryLand datasets. Recently, in [67], they slightly modify their previous work that outperformed even *C3D* a deep learning approach. In their previous work on *DSR*, simple color histograms are used to capture chromatic information. Whereas, in this approach the model

incorporates and shows improved performance when moving from histogram to mean and variance based color representations.

2.1.2.2 Deep Learning for DSR

DL has been many times used for scene recognition from still images. However, very little work has been done for dynamic scene recognition using *DL*. As previously discussed in Sec. 2.1.1.2, D. Tran et al. [23] propose a model that learn spatiotemporal features from videos using 3D *CNN*. In which they train a model with Sports 1Million dataset, and than use that trained model to extract features at fc6 layer from other datasets. These fc6 layer activations are averaged to form a 4096-dim video descriptor. L2-normalization is applied on that feature vector. These normalized feature vectors are called C3D features. They apply the trained network on YUPENN and Mary-Land dataset. Its *C3D* features are classified with a linear *SVM* that shows better result than many traditional hand-crafted approaches as well as ImageNet architecture.

Recently, A. Gangopadhyaya et al. [74] propose a model that analyze the performance of statistical aggregation (*SA*) techniques on various pre-trained *CNN* models to address *DSR*. It works by extracting CNN activation features for a number of frames in a video and then use an aggregation scheme in order to find robust feature descriptor for each video. A 4096 dimensional vector is extracted from each frame which contains rich spatial information and represents a single frame in the given video. All of these feature vectors are combined in one to capture the temporal statistics of the spatial features. It is done to induce a certain degree of temporal invariance and extract temporally un-ordered characteristics. They consider two strategies to aggregate these spatial descriptors temporally, i.e. statistical aggregation and vector of locally aggregated descriptors (VLAD). In statistical aggregation, they used mean, standard deviation, skewness, kurtosis, and Max. Whereas, VLAD is the popular vertical pooling strategy developed after bag-of-words feature en-

coding. In VLAD the features are classified by a linear *SVM* in the end. They have 3 pretrained networks, i.e. i) trained with AlexNet (1000 categories), ii) with Places Dataset (having 205 scene categories), iii) and Hybrid network (trained with combination of 205 Places dataset categories and 978 object categories from ILSVRC12 dataset). These trained networks are used to extract features from YUPENN and MaryLand datasets. These fc6 or fc7 layers features for certain number of frames are chosen from the video. The final feature vector for each video is calculated by performing temporal aggregation using statistical moments, max pooling or vectorial pooling. The feature vector is given to a linear *SVM* to classify each video based on one-vs-all classification criteria. Their approach resulted in optimal result for both YUPENN and MaryLand datasets.

Based on the literature review, we have concluded that the 3D structure in deep models such as *3DCNN* and *C3D* have great impact on spatiotemporal feature learning. This 3D structure give a generalization power to the models. Therefore, our work is an inspiration of these models. We try to propose an architecture that may learn features from input up to the output without incorporating any sophisticated pre-processing steps.

2.2 History of Pyramidal Neural Networks (PNN)

In *NNs*, parameter learning algorithm or model is solely not important. Its combination with appropriate prior knowledge of the architecture and connections play an important role for reducing the number of learning parameters, network complexity and execution time. However, sometime question of accuracy is not raised while comparing networks with fewer parameters vs. more parameters. Excessive and redundant number of features does affect the performance of a network. Feature reduction/selection techniques like *PCA* [75], *LDA* [76], and *ICA* [77], etc. are commonly used to select most discriminative features at an extra cost. However, image pyramids

(*IP*) can do that role more efficiently than the statistical tools. Both *IPs* and *NNs* have parallel topology.

IPs have shown to be an effective and discriminative data processing structure for images, which reduces redundant and non-discriminative data exponentially. A simple *IP* consists of several levels of filtered and sampled images stacked over each other containing original image at the bottom followed by next bigger filtered image and so on. The filtering can be linear or nonlinear, e.g. Gaussian or Max etc. depending on the image and the scenario of the problem. This structure helps algorithms to reduce their time complexity from $O(logn^2)$ to $O(logn)$. The content and behavior of *IPs*, i.e. cell, level, neighborhood, and processing performed by those cells can be considered an equivalent terminology in *NN* as neuron, layer, receptive field, activation functions, respectively. These make them similar and favorable to early perceptron's and *NNs*. Especially, the structure, i.e. regular structure or irregular structure gave an intuition for the receptive field concept in later *NN* models, i.e. *Neocognitron*, *CNN* etc. [11, 78, 33].

According to H. Bischof and W. G. Kropatsch [79], any pyramid can be stored in graphs with $'n'$ neighborhood graphs and $'n-1'$ vertical graphs, whereas in case of regular pyramids one doesn't need to save all graphs. Based on these understandings one can convert and represent any regular pyramid to a *NN* model; however the problem of shift variance will remain as it is. Further, they describe how one can interchangeably use these concepts in order to propose further *PNN*. They propose few of the early supervised and unsupervised *PNN*, e.g. supervised ($2 \times 2/2$ curve pyramid, neural network pyramid) and unsupervised distributed curve pyramid. Despite similar structure as *NN*, the question remains for selecting suitable learning algorithms for the pyramidal architectures in order to tackle local minima problem as well as faster convergence. Besides mostly used supervised learning techniques, a more suitable learning algorithm could be an unsupervised to extract the hidden capabilities of *PNN*

by tackling slow learning and bad scaling characteristics. Over the passage of last four decades, several models are presented. Some of the famous *PNNs* are given below, which later somehow became the base for most *DL* architectures.

Starting from the base idea of 1960's, Hubel and Wiesel divided brain cells of the visual cortex region in three classes, i.e. simple, complex and hyper-complex cells. In their hypothesis, visual information use to be processed hierarchically from simple cells to complex cells and complex cells to lower-order hyper-complex cells and further to high-order hyper-complex cells as shown in Fig. 2.1. In the coming sections we discuss some of the early *PNN* as well as *PyraNet*.

2.2.1 Neocognitron

Hubel and Wiesel further put forward that relation between lower-order complex cells to higher-order hyper-complex cells seems similar to the relation between simple cells to complex cells. This is the idea that becomes the seed for Neocognitron model proposed by Fukushima [33]. However, now neurophysiologists are not giving any importance to the idea of classifying hypercomplex cells into lower-order and higher-order. Similarly, the early Neocognitron is composed of two major cells, i.e. simple and complex in short S-cells and C-cells cascaded after each other which does feature extraction from low level like edges to high level like higher-order features specific to patterns, are extracted and then finally classified in object categories.

This network is an iterative, self-organizing and competitive learning model, where all output units responds to the input image in response of any activity. The neuron with maximum value is selected and a strengthening signal is sent down to the receptive field neurons in the layer below, whereas other cells decline. This strengthens only those signals which have specific meaning, i.e. discriminative features of an object while neglecting the rest. Further, after recognizing one

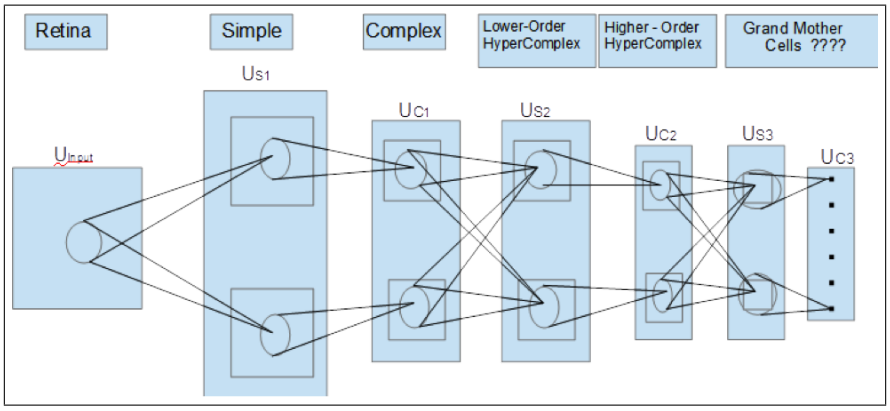


Figure 2.1: Fukushima Neocognitron, where on top of each layer its relation is shown with Hubel and Wiesel Classical hypothesis.

pattern, the system automatically transfers towards another object and tries to learn its pattern by interrupting the previously 'on' signals by a decay term that causes a gradual decrease in their activities letting the previously suppressed cells to win in order to learn any other pattern in the image. In this way, for the first time kernels learn the shape of fully or partially occluded objects through ensembles of non-connected regions of attentions in input patterns. However, it is quite complex for the systems at that time and needed a lot of time for convergence [11, 33].

The same model is still under research for further enhancement with modification in network structures and learning algorithms. However, its proper pyramidal structure has been changed due to changing number of maps in each layer. Fukushima proposes a newer version of Neocognitron [80], in which several new concepts and layers are introduced for improvement of recognition rate. These contributions includes: a) Inhibition effect from surrounding neurons between S-cells to C-cells, b) new layer for extracting contrast from the input maps followed by edge detection layer, c) modification in learning strategies like 1) self-organizing of line-extraction cells, 2) unsupervised learning in the middle layers with usage of thresholding with larger values for extracting edges, 3) supervised competitive learn-

ing at the final layer where the threshold value is kept low based on their previous experience that results in more training but less recognition time due to less redundant reference vectors inside the class borders, d) staggered arrangement of S-cells and C-cells for removal of accessory circuits present in the previous versions that results in simplification of the network architecture as well as improvement of recognition rate. However, the model has the deficiency that one has to check performance for different combination of prior learned threshold values and their combination.

Recently, K. Fukushima proposes a newer version of Neocognitron in which he proposed 'add-if-silent' learning rule [81]. Based on which if all S-Cells in a receptive field are silent despite of non-zero training stimulus, than a new S-Cell generates and adds to the network that learns the presented training pattern. If there exist more silent receptive fields, the same procedure repeats until the whole area cover with non-silent S-Cells. Further, two other learning rules are used, i.e. 'winner-take-all' and 'winner-kill-loser' rule. The advantage of using 'add-if-silent' and 'winner-kill-loser' rule is not only that it achieve higher recognition rate compare to other like 'winner-take-all' rule, but to stabilize the network by making it monotonically increasing the number of cell-planes and stopping it when the reference vectors cover the multidimensional feature space. This also reduces the computational cost of Neocognitron. In addition, as in many neural networks and previous versions of Neocognitron, averaging is used at C-Cells, which provide spatially blurred effect in the succeeding cell-planes that is also called spatial pooling. This provide smoothing affect to additive random noise present at S-Cells. Instead of averaging, this time Root-Mean-Square is used which is not very different than averaging, but it increase the recognition rate by providing reduction in fluctuation from response of C-cells caused by spatial shift of a stimulus feature. At highest layer they show that interpolating-vector can greatly increase the recognition rate in combination to the above discussed work.

The reason to discuss Neocognitron models is to highlight and to give respect to Fukishima as initiator of many ideas being used in todays *NN*.

2.2.2 Honavar and Uhr Model

V. Honavar and L. Uhr in 1989 strictly followed the idea of Hubel and Weasel visual cortex architecture in their pyramidal network [11, 78]. This network combined the idea of pyramid structure with *MLP* architecture and back-propagation (*BP*) learning rule. In addition, they introduce prior knowledge architecture by specifying a receptive field instead of fully connected neurons. This receptive field concept reduced the spatial resolution exponentially. Whereas, the learning algorithm repeatedly updates weights to correct the output of each neuron.

A translated version of the input image or lower layer map is cascaded over current layer map, in which the pattern is at a distance of eight pixels from the image centre. This network is composed of six layers and is dependent on image size. The larger the image, larger the number of hidden layers and number of weights. However, it is tested on a very small and limited dataset. Than the idea of *CNN* came with weight sharing concept that is discussed in later section.

2.2.3 McQuoid Model

McQuoids [82] in 1993 propose a four layer *PNN* to recognize even multiple digits at a time based on the idea of visual system of our brain, i.e. retina as input image followed by three neuronal layers. The first layer results from planes of ensembles, each aligned above one another, creating a column of ensembles consisting of 5×5 grid of neurons connected to input image directly using same shared weights. There is only one column for each input neuron belonging to an input receptive field of 7×7 neurons, connected by shared weights that

provide translation invariance. The same structure is followed by second layer with 5×5 receptive field, however, layer 3 is made of neurons, one for each class and it worked as a monitor for verifying the presence of continuous patterns of activation and can be realized by any algorithm even other than connectionist.

This network for an image of size 24×24 and 3 areas consist of 19882 neurons with 948735 links. Online approach for learning is adapted to recognize the binary images having digits 0-4 and their combination with impulsive noise but no rotation and translation invariance. Its capabilities are highly limited to rotation and translation invariance. In addition, it also behave as an associative memory that memories the accomplish task with large number of weights that can be achieved with less number of associative memories by other tasks.

2.2.4 Cantoni & Petrosino Model

V. Cantoni and A. Petrosino [11] in 2002 propose a *PNN* architecture that is somewhat similar to Honavar et al.[78] but strict compared to *CNN*, Neocognitron and some other models. Because the resolution is reduced continuously at each higher layer and transfer of information is refined at each step similar to visual cortex system. Compare to Honavar et al. the base layers are composed of multi-resolution analysis of the input image, followed by a structured receptive field *NN* approach for classification as shown in Fig. 2.2. One of core idea is to limit the number of weights up to higher levels instead of using and learning till the input image. In deeper network, local gradients in the *BP* becomes smaller and smaller and finally almost becomes negligible in the layer closer to the input pixels. Therefore, they used free parameters for classification only.

The feature map given to this network is extracted by a system inspired from the human visual system that recognize things in a scale independent way, i.e. multi-resolution approach. Multi-resolution approach at the base preserves the spatial localization and spatial

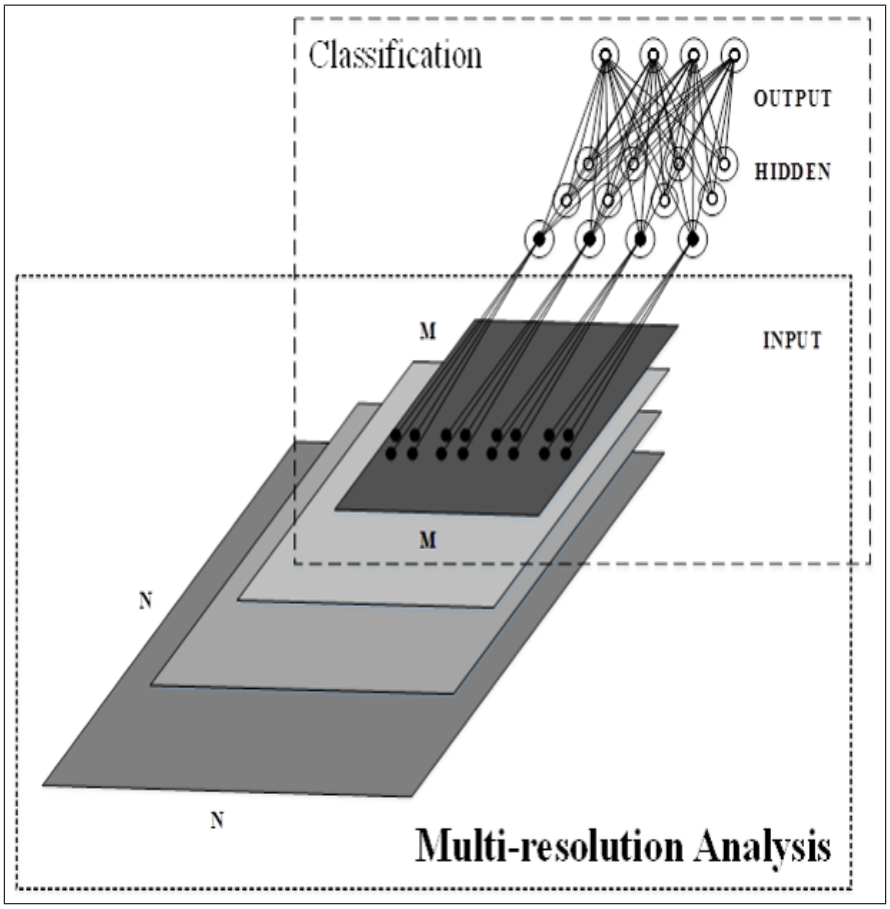


Figure 2.2: Cantoni and Petrosino Model.

frequency information because the functions and variables used are based on local information of the data which reduces the frequency content with the help of local low pass filtering e.g. Laplacian pyramids in this case. The stopping criteria for reduction of the features from the input in the multi-resolution analysis is based on the prefixed parameter, i.e. $log(M)$ where *'M'* is less than *'N'*, the size of the input image. But this *'M'* can be kept totally fixed in order to increase the number of layers in the case of high number of classes in the input images. This is highly correlated to the information content of the images provided to the system. The map at layer 'M' of the Laplacian pyramid is given to the classification stage which is followed by a structured hidden layer, i.e. a neuron at $l + 1$ layer

that is connected to 4×4 neurons at layer $l = m$, and then neurons at layer $l + 2$ are connected with 2×2 neurons of layer $l + 1$.

This structure is invariant to rotation, shift, and noise. But it is also successful to identify the partially occluded patterns in the image due to its similarity with visual attention concepts, i.e. scanning mechanism for attaining selective attention. It scans from left to right and top to bottom until encounter a neuron from the pattern body which they identified by transitioning of the pixel from white to black, and then setting the receptive field from that neuron. Their results are $4 - 5\%$ better in classification than Hanover and Uhr's system at the same dataset of tools and letters having less parameters.

2.2.5 PyraNet Model

S. L Phung and A. Bouzerdoum [14] propose multilayered architecture inspired by *IPs* and receptive fields concept of *NN* for classification of visual patterns. Their *PNN* is named as (*PyraNet*). It is quite similar to V. Cantoni and A. Petrosino *PNN* [11] in pyramidal structure but the only difference is that *PyraNet* model in pyramidal layers learns coefficients of the receptive fields adaptively. Whereas, in V. Cantoni and A. Petrosino *PNN* coefficients of the low pass filters are fixed. However, higher layers are learned.

PyraNet is also similar in structure to *CNN*. *PyraNet* and *CNN* both follow same *BP* technique for learning parameters with cross-entropy loss function. However, this *BP* technique is modified for the new weight parameters scheme as shown in coming subsection. 2.2.5.3. Further, it differs from *CNN* in three main aspects. First, *PyraNet* has no pooling layers for reduction of dimensions, rather the dimensions are reduced by the stride of the kernel at each layer. Similar reduction procedure has been adopted now in new deep CNN models [83]. Secondly, it does not perform convolution rather weighted sum (*WS*) operation or correlation over the receptive field. Finally, in weight sharing concept, where *CNN* has a kernel to fil-

ter the whole image and may extract low level features like edges. Whereas, *PyraNet* is not just limited to low level at a specific position, i.e. weights are not in the form of a kernel that slides over the whole image, rather each output neuron has a local unique kernel specifically assigned to it. These kernels are based on input neurons in a receptive field and their corresponding receptive field (weights) in a weight matrix. This results in unique locally connected kernel for each output neuron. However, the locally connected kernels in *PyraNet* are up to some extent shared for another neuron unlike the locally connected layers in *CNN* [84].

PyraNet used five different training methods for optimization and reduction of error, i.e. three of them are first order optimization techniques (Gradient Descent (GD), Gradient Descent with Momentum and variable learning rate ($GDMV$), and Resilient BP ($RPROP$)), fourth is Conjugate Gradient (CG) that is in the middle of first and second order, while fifth is Leveberg-Marquadrt (LM) which is from the second order methods. They examine the convergence speed and computational load in order to handle large datasets with these training algorithms for *PyraNet*. To compare the speed and load of different networks, they propose a unit known as gradient descent epoch time unit ($gdeu$) where one $gdeu$ is the mean of the time taken for performing one GD training epoch on a fixed training set and a fixed number of size for the networks.

2.2.5.1 Architecture

Fig. 2.4 shows basic architecture of a PyraNet. It consists of two types of several processing layers. At the base, it comprise of 2-Dimensional pyramidal layers that does dimensionality reduction and feature extraction. Whereas, on the top, 1-Dimensional fully connected layers that perform classification of patterns. First 2D-pyramidal layer is directly extracting features from real input image that are refined and provide as input map to the higher one or more consecutive 2D-pyramidal layers. Each neuron at higher layer is the

output of weighted sum of a neurons (shown in Eq. 2.1) in a receptive field (set of neurons calculated using Eq. 2.2) at lower layer with adjacent weight parameters in weight matrix for that layer. Each neuron is passed through an activation function, e.g. sigmoid or hyperbolic tangent to produce output map. The output of last pyramidal layer is connected to fully connected 1-D layers. These can be further connected to other 1-D layers in case of complex problems. Whereas, output of last fully connected layer is considered the output to represent the given input.

2.2.5.2 Weighting Scheme

An important part in popular convolutional deep models is their weight-sharing concept that give an edge over other *NN* models. This property reduces large amount of learning parameters. However, increases burden on those fewer parameters. Fig. 2.3 shows the weighting scheme adopted in *PyraNet*. Each neuron has its own unique weight. This unique weight scheme results in weight matrix that have same size as the input image or feature map at lower layer. And at the time of computation, each output neuron gets a unique kernel based on calculated receptive field. This behave like position oriented kernels. These are learned using updated back-propagation technique and stochastic mini-batch gradient decent approach.

The output of 2D-Neuron in a pyramidal layer is calculated by:

$$y^{l_n}_{(u,v)} = f_l \left(\sum_{(i,j) \in R^{l_n}_{(u,v)}} W^{l_n}_{(i,j)} \; x^{l_{n-1}}_{(i,j)} + b^{l_n}_{(i,j)} \right) \tag{2.1}$$

Where $l = 1, 2, 3..L_p$ and if $l = 1$ than it represents real input. Further, $R^{l_n}_{(u,v)}$ is a 2x2 receptive field of neuron (u, v) at level l given by

$$R^{l}_{u,v} = \begin{cases} (i,j) \,|\, (u-1) \; + 1 \leq i \leq (u-1) + \; r_l; \\ \quad (v-1) + 1 \; \leq j \; \leq (v-1) + \; r_l \; ; \end{cases} \tag{2.2}$$

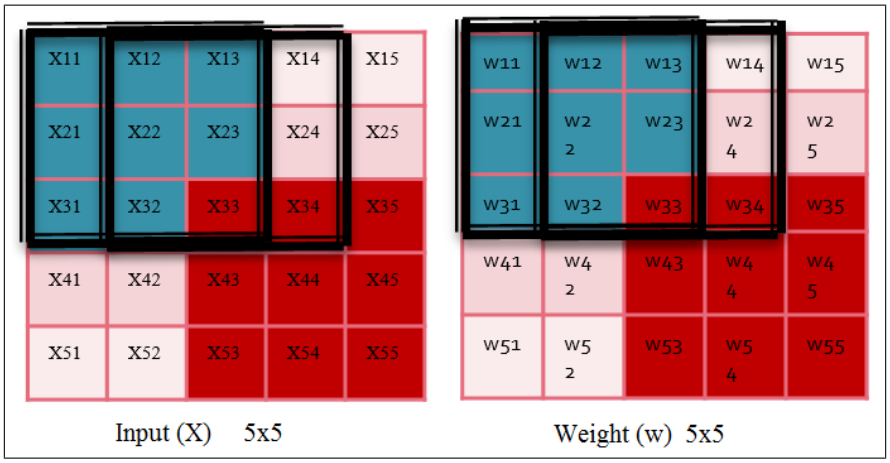


Figure 2.3: PyraNet Kernel Architecture

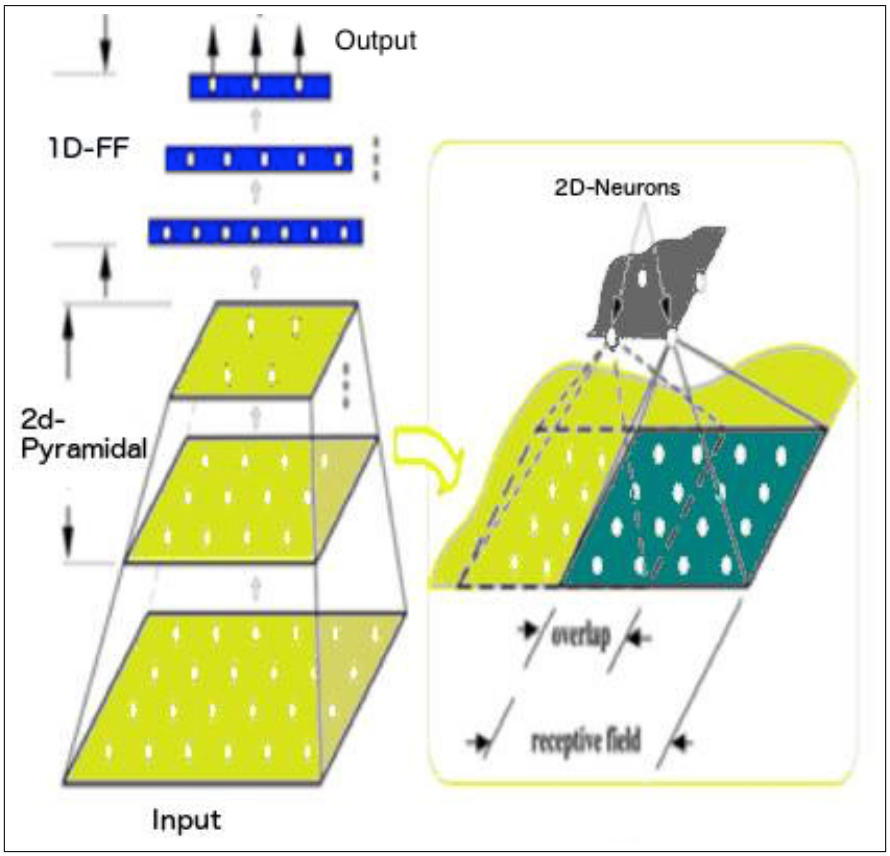


Figure 2.4: PyraNet Model.

Size of receptive field is represented by r_l, gap and overlap of a kernel in pyramidal layer is represented by g_l and o_l, respectively. The gap factor is calculated by subtracting o_l from r_l. The output of $y^{l_p}_{(u,v)}$ of the last pyramidal layer is rearranged by vectorizing the 2d matrix in a 1-D matrix by arranging each column after the other in a vector form as shown in Eq. 2.3. This 1-D vector is given as input to the upper 1-D fully connected layer.

$$\begin{aligned} y^{L_p}_{(u,v)} \; | u = 1, 2, \ldots H_{L_p}, \; v = 1, 2, \ldots W_{L_p} \\ - > \left\{ y^{L_p}_m | m = 1, 2 \ldots N_{L_p} \right\} \end{aligned} \tag{2.3}$$

The last 2-D and 1-D layer are used in same manner like a fully connected layer of a *MLP*. To calculate output for each neuron in 1-D layer, $W^l_{m,n}$ be the synaptic weight from neuron $'m'$ in layer $'l-1'$, to neuron $'n'$ in layer $'l'$. b^l_n represents the bias for neuron $'n'$ in the layer $'l'$. The output of this 1-D neuron is given by Eq. 2.4. $l = L$ represents output layer of the network.

$$y^l_n = \; f_l \; \left(\sum_{m=1}^{M_{l-1}} W^l_{m,n} \; y^{l-1}_m + \; b^l_n \right) \tag{2.4}$$

2.2.5.3 Training Model

PyraNet model adoptes two approaches for training that are generally used. First approach is to use network output directly while in the second approach they have utilize the posterior probabilities of class membership. For this reason, they have used mean square error (*MSE*) shown in Eq. 2.5 for first approach, and cross-entropy (*CE*) as shown in Eq. 2.6 for the second as error function, respectively.

$$E_{MSE}(w) = \frac{1}{K \times N_L} \; \sum_{k=1}^{K} \sum_{n=1}^{N_L} \left| y^{L,k}_n - d^k_n \right|^2 \tag{2.5}$$

$$E_{CE} \; (W) = - \; \sum_{k=1}^{K} \sum_{n=1}^{N_L} d^k_n \; ln p^k_n \tag{2.6}$$

Where d_n^k represents 1 for selected class otherwise 0 as shown below:

$$d_n^k = \begin{cases} 1 \; if \; desired \; class \\ 0 \; otherwise \end{cases}$$

And p_n^k is a posterior probability in result of applying softmax function on output.

$$p_n^k = \exp\left(y_n^{L,k}\right) \Big/ \sum_{i=1}^{N_L} \exp\left(y_i^{L,k}\right), \qquad n = 1, 2, 3 \ldots N_L \tag{2.7}$$

It use an updated version of back-propagation technique to update the weight parameters for the network.

Calculating Error gradients

The weight gradients are updated in two steps. First, it calculate error sensitives for each neuron at each layer, and then error gradients that are subtracted from old weights.

Step1 Error Sensitivities:

These error sensitivities are partial derivatives of error at output layer (calculated with *MSE* or *CE*) with respect to weighted sum input. However, there is slight difference between 1D and 2D neurons. This difference is because of the connection difference at pyramidal layer and fully connected layer. For 1D layer the error sensitivity is calculated by Eq. 2.8 and for 2D layer by Eq. 2.9.

$$\delta_n^{l,k} = \frac{\partial E}{\partial s_n^{l,k}} \qquad where \; l > L_p \tag{2.8}$$

$$\delta_{u,v}^{l,k} = \frac{\partial E}{\partial s_{u,v}^{l,k}} \qquad where \; l < L_p \tag{2.9}$$

The calculation of error sensitivity for output layer, 1D layer and 2D layers is different. Therefore, they calculated it with the help of given equations. At output layer;

$$\delta_n^{l,k} = \frac{2}{K * N_L} \; e_n^k \; f_L^{'}\left(S_n^{L,k}\right) \quad where \;\; n = 1, 2 \ldots N_L \tag{2.10}$$

Other 1-D layers ($L_p < l < L$)

$$\delta_n^{l,k} = f_l^{'}\left(S_n^{l,k}\right) \sum_{m=1}^{N_{l+1}} \delta_m^{l+1,k} \; W_{m,n}^{l+1} \tag{2.11}$$

For last pyramidal layer:- ($l = L_p$)

$$(\delta_n^{L_p,k}, \quad n = 1, 2, \ldots N_{L_p}) -> \delta_{u,v}^{L_p,k} \tag{2.12}$$

where $u = 1, 2, \ldots H_{L_p}$ and $v = 1, 2, \ldots W_{L_p}$. While, in pyramidal 2D layers error sensitivity is calculated by:

$$\frac{\partial E}{\partial S_{u,v}^{l,k}} = \; f_l^{'}\left(S_{u,v}^{l,k}\right) \; \left[w_{u,v}^{l+1} * \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j=j_{high}} \delta_{i,j}^{l+1,k} \right] \tag{2.13}$$

Where as the $i_{low}, i_{high}, j_{low}$ and j_{high} are calculated by:

$$i_{low} = \left\lceil \frac{u - r_{l+1}}{g_{l+1}} \right\rceil + 1$$

$$i_{high} = \left\lfloor \frac{u - 1}{g_{l+1}} \right\rfloor + 1$$

$$j_{low} = \left\lceil \frac{v - r_{l+1}}{g_{l+1}} \right\rceil + 1$$

$$j_{high} = \left\lfloor \frac{v - 1}{g_{l+1}} \right\rfloor + 1$$

Step2 Error Gradients (Weight and Bias gradients): Now after calculating error sensitivity for each neuron, the weight gradients are calculated. The error sensitivity associated with specific input neurons are multiplied and summed to calculate the fractional value to be subtracted from the previous weights. To calculate error

gradient for 1D layer ($L_p < l < L$), Eq. 2.14 is used.

$$w_{m,n}^{l} = \frac{\partial E}{\partial w_{m,n}^{l}} = \sum_{k=1}^{K} \delta_{n}^{l,k} \; y_{m}^{l-1,k} \tag{2.14}$$

where $m = 1, 2, \ldots . N_{l-1}$, and $n = 1, 2, \ldots . N_{l}$. Now to calculate bias of those layers it use Eq. 2.15.

$$b_{n}^{l} = \frac{\partial E}{\partial b_{n}^{l}} = \sum_{k=1}^{K} \delta_{n}^{l,k} \tag{2.15}$$

Whereas at pyramidal layers ($l \leq L_p$), Eq. 2.16 is utilized for weight gradients and Eq. 2.17 for calculating bias's of that layer.

$$w_{i,j}^{l,k} = \frac{\partial E}{\partial w_{i,j}^{l,k}} = \sum_{k=1}^{K} \left\{ y_{i,j}^{l-1,k} * \sum_{u=u_{low}}^{u_{high}} \sum_{v=v_{low}}^{v_{high}} \delta_{u,v}^{l,k} \right\} \tag{2.16}$$

$$b_{i,j}^{l,k} = \frac{\partial E}{\partial b_{i,j}^{l,k}} = \sum_{k=1}^{K} \delta_{u,v}^{l,k} \quad Where \; u = 1, 2, \ldots H_l \; and \; v = 1, 2 \ldots W_l \tag{2.17}$$

Whereas, the neuron in receptive field of pyramidal layers is calculated by:

$$u_{low} = \left\lceil \frac{i - r_l}{g_l} \right\rceil + 1$$

$$u_{high} = \left\lfloor \frac{i - 1}{g_l} \right\rfloor + 1$$

$$v_{low} = \left\lceil \frac{j - r_l}{g_l} \right\rceil + 1$$

$$v_{high} = \left\lfloor \frac{j - 1}{g_l} \right\rfloor + 1$$

2.2.5.4 Experiments & Results

The performance of *PyraNet* is compared on five networks for speed and accuracy. Experiments show that *GD*, *GDMV* are the slowest by having 26900 and 20700 *gdeu* compared to 65.5, 735.5, and 1035.5 *gdeu* of *LM*, *CG*, and *RPROP*, respectively. Similarly, the performances of these train algorithms for varying training epochs up to 2000 show that *RPROP*, *CG*, *LM* algorithms dominate by achieving highest classification rates, i.e. 97.3%, 97.2% and 97.5%. Whereas, *GD* and *GDMV* are last by correctly classifying only 96.5% and 96.6%. However, *GD* and *GDMV* need more time to find good trained parameters and classify accurately. Similarly, they also show that training a *PyraNet* with *MSE* or *CE* error functions had no significant difference in performance. *PyraNet* with *RPROP* and *CG* have acceptable training speed and need less memory. However, with *LM* algorithm it is fast in convergence but needs more memory during execution. *PyraNet* achieved 96.3% accuracy similar to *SVM* for gender recognition and 5% more than *CNN* with same input size images of *FERET* dataset.

2.2.6 I-PyraNet Model

Bruno et al. [85] extended *PyraNet* by introducing the concept of inhibitory fields. Inhibitory fields in addition to the receptive fields can send not only excitatory stimulus but also inhibitory stimulus to the neurons in the posterior layers. Inhibitory fields concept is introduced by G. Rizzolti et al. [86], which states that in addition to receptive field stimulus there exist other stimulus outside the receptive field which affect the neuron known as inhibitory stimulus more commonly known as non-classical receptive field. These kinds of stimulus are found in visual cortex of macaque monkey and have some influence on decision-making.

The architecture is similar to *PyraNet*, but the only difference is the subtraction of the weighted sum of inhibitory field from the

weighted sum of the receptive field before activation of a neuron. In pre-processing, it use Gabor filtered images as input to the *PyraNet* and applied histogram equalization to achieve better accuracy. The training is done with same *BP* algorithm, whereas they used *CE* function to minimize the error. *I-PyraNet* is tested on MIT CBCL dataset for face detection and compared it with *SVM* and *PyraNet*. *I-PyraNet* achieves better results than *PyraNet*, but failed to get better accuracy than *SVM*. However, *I-PyraNet* is 175 times faster than SVM, which can motivate its use in some applications for future use.

In [87], *I-PyraNet* and non-classical receptive field approach is used for satellite image segmentation. They propose a model for segmentation and classification with receptive field ($SCRF$). The reason behind using *I-PyraNet* is to perform image segmentation and classification based on not only the value of the pixel, but also by its neighbor pixels to define a pixel class. The image is divided in two different sizes of receptive field with overlap, and each sub-image is classified in class and non-class pixels with the highest probability that is calculated through supervised classifier, i.e. *I-PyraNet*. *SCRF* shows good results for segmentation compare to *k-NN* and *PyraNet*. From the experiments on segmentation of satellite images for forest detection, their *SCRF* with *I-PyraNet* can achieve error rate from 10% to 2.49% compare to manually segmenting the images, and are better than other stated in literature in-terms of having less error rate and processing speed.

Bruno et al. [88] proposed an extension of the same concept, with slight modification of inhibition of the neuron even at the same layer. They show that the *LIPNet+SCRF* not only shows comparable results to *SVM* but it consumes less memory by saving less parameters and have better and faster results than *SVM* if several results are averaged for the same data. Recently, they introduced a new model called variable pyramidal neural network with evolutionary algorithm (*VPNN* or *VPNN-EA*) [89]. In this model, rather using

fixed receptive field, they use variable receptive fields determined by an evolutionary algorithm. They report better performance for detecting faces in images than *PyraNet* and *LIPNET*.

2.2.7 LCNP Model

R. Uetz and S. Behnke [90] propose a system known as locally-connected neural pyramid (*LCNP*) due to having fixed receptive field with unique connection weights for each neuron. It results in large amount of free parameters compared to CNN. This model follows the same idea as [11, 14] for receptive field and having unique connection for each weight like *PyraNet* [14]. The justification for using such weights structure given by R. Uetz and S. Behnke is its biologically motivation, i.e. same connection structure is massively used by *BNN*.

Similarly, it does not follow the idea of weight sharing as in *CNN* [25]. But it does follow them in two points, increasing the feature maps as the layer increases and a fixed receptive field with weights from the lower layer to generate a neuron at upper layer. Their network consist of at least one map in a layer, however, maps increases as the layers increase, i.e. follows *CNN* type architecture of providing many kernels to produce many maps in each layer. Last output layer is a fully connected layer, where each neuron represents a class. Supervised learning using 'plain-vanilla' *BP* of error is used. To control the learning, it uses fixed but different learning rate value for pyramidal layers than the output layer. In addition, *LNCP* use the same input layer maps in all layers after subsampling them to the size of each layer in advance to make them of equal size of their neighboring maps of the current layer. This is done because lower layer edge filters provide fine-grained edges compare to high-level layers that provide coarse-grained edges.

They implemented a parallel version code for *PyraNet* using GPU support, which is 82 times faster than CPU implementation. To evaluate the performance, it is tested on three datasets, i.e. LabelMe,

MNIST and NORB. They achieve good results for NORB better than all other state-of-art techniques, i.e. error rate less than 0.05% on training and 2.87% on testing dataset. However, for MNIST dataset it is not as better as *CNN*, i.e. 0.03% for training and 0.76 for testing. LabelMe is a new dataset that they introduce on which they achieved 3.77% error rate for training and 16.27% for testing.

2.2.8 Conclusion

Although, the techniques described here are not all specifically for *AR*, but there are three reasons to explain these old *NN* techniques. First, we want to highlight some of those who worked in the start for providing building blocks of *DL* concepts. Second, we want to show the impact of pyramidal architectures on *NN* and whether it can be improved further by following strict pyramidal architecture or not. In the end, as one of the main building blocks in *NN* is weight parameter that play vital role in performance, therefore we have mentioned the weighting scheme proposed in *PyraNet*.

In early days, most of them tried to reduce weight parameters due to large number and high time complexity in learning, but it can result in reduction of accuracy. In addition, fully connected connections result in huge amount of learning parameters. Training needs large amount of training data to learn better and avoid over fitting. Otherwise, this results in less testing accuracy. Weight sharing concept solve some of the previously discussed issues. But what if one use combination of these two concepts, can it provide some enhancement, i.e. individual weights for each neuron and receptive field concept rather than fully connected as in previously discussed research work [11, 14, 87, 85, 81, 90, 88, 25].

From discussion of previous few papers, a positive sign can be depicted for utilizing such architecture for better performances. Nowadays, with latest technology and powerful computers, researchers bypassed being worried for large number of parameters compare to scenario where cent percent accuracy is more important. Therefore,

one can observe and further investigate performance of this middle approach, i.e. individual weights for each neuron at lower level that can result in different filters for each receptive field similar to [14, 87, 85, 81, 90, 88, 25], that may provide more absorbing capabilities to weights. The weight parameters will be more than traditional *CNN* have but still less than the recent far deeper networks.

The next section describes another most known and usable model of a pyramidal neural network approach, i.e. *CNN*. Initially, it not only have pyramidal structure but also utilizes receptive field concept more efficiently that helped in reduction of parameters as well as increasing performance. This model is reborn recently in [12]. In the next Sec. 2.3, we discuss some of the main models of *CNN* that brought revolution in *CV* and *ML*.

2.3 Some Variants of CNN since LENET5

In the era of Neocognitron and Honavar & Uhr systems, Y. Lecun et al. propose *CNN* with emphasizes on weight sharing concept along with the existing idea of receptive fields. This not only reduces the complexity and number of weights but also provides the start of using images as input instead of providing refined extracted and selected features to the neural systems. Recently G. Hinton and A. Krizhevsky et al. [12] propose a new *CNN* model similar to Lecun *LENET5* but with more hidden layers and with few new amendments. The name 'Deep' coined due to increasing hidden layers in *CNN*. The performance of *DL* is day by day improving compare to other state-of-art techniques like *BoVW* and other *CV* and *ML* techniques.

There are models even before A. Krizhevsky deep *ConvNet* that have many hidden layer like some discussed in previous section. Than what is the reason behind its success? Is it really due to going deeper by increasing hidden layers, or because of going wider by increasing kernels and maps? Is it because of pooling? Or is it because of the increase in labeled training and testing data? Or is it due to powerful computer and technology? Or due to introducing hybrid networks that resulted due to combination of new layers like pooling, or steps like sharing, receptive field, activation functions, normalizations, inceptions, or especially due to GPU? Since last few years enormous work have been done on *CNN* and other similar *DL* models, e.g. Restricted Boltzman Machine *RBM*, Recursive NN *RNN*, Autoencoders etc. [91]. However, here we discuss and focus only on those models that are build on the basis of *LENET5* convolutional network, Dropouts, RELU etc., that provides some novelty and great impact in the area of *OR*. In coming subsections, we explain few *CNN* models.

2.3.1 LENET5

CNN is introduced by Y. Lecun et al. [25] for character recognition problem. They introduce the concept of weight sharing and receptive field in order to reduce the number of parameters in the network. Its first big success is on hand written digit recognition. *LENET5* implicitly extracts related features to produce a map on a current layer using shared kernel (weights), applied on the locally connected area from the previous layers. This work similar to the pyramid approach and step not only reduces spatial resolution of the feature map but also provide some shift and distortion invariance along with enhanced edges, corners or endpoints on the output map. In addition, due to sharing of weights the number of learning parameters reduces enormously. After several step of convolution and subsampling, finally the features produced are vectorized and provided to a fully connected layer for classification using a softmax logistic regressor.

The topology of the *LENET5* contains 8 layers including input and output, i.e. three convolutional layers where first two are followed by subsampling layers that provides translation invariance. The third one is a special convolution layer fully connected to a 120 neurons that are further connected to another fully connected 84 neurons like a multi-layer perceptron for classification. Finally, these 84 neurons are computed for output using Euclidian radial basis function (RBF). For learning the training parameters, *BP* is used. However, to speed up the network they adopt stochastic second order derivative approach of the *BP*, i.e. Stochastic Diagonal Levenberg Marquardt Method, where they learn a learning rate value for each weight parameter instead of hard coded small value for learning rate. *LENET5* recorded about 0.35% error rate just after 19 epochs and that trained parameters resulted in 0.95% of testing error rate. The key aspects in this model gave intuition to researchers for utilizing and modifying *CNN* for other fields in *CV* and *ML* such as Speech,

medical, biometrics, etc. with slight modification of input maps, mapping schemes, output neurons, etc.

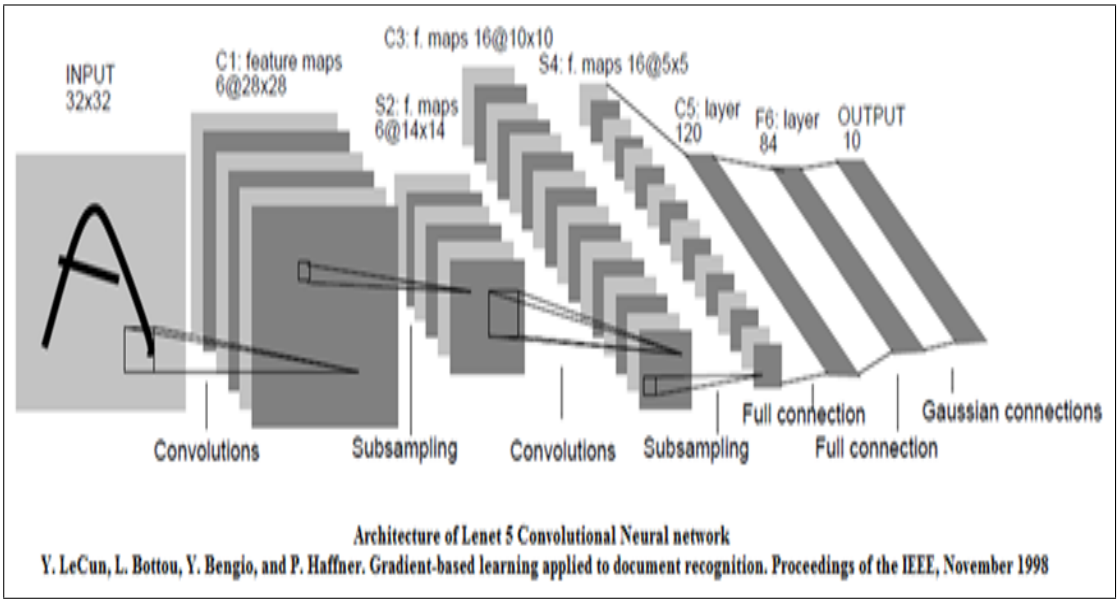


Figure 2.5: LENET5 Model

2.3.2 AlexNet

Since couple of decades, *CNN* is focus area of research for Y. Lecun and G. Hinton, but in 2012 G. Hinton with his student A. Krizhevsky proposed a new model [12], i.e. *AlexNet* for object recognition that revolutionaries *CNN* and hence coined the word *DL*. *AlexNet* [12] gave rebirth to *CNN* and shows great results in ImageNet ILSVRC 2011 and 2012 classification benchmark [29]. In their successful model, the new key factors are: efficient implementation of distributed powerful GPU based implementation on different layers of the model that made the training of very large datasets practical for researchers, introduction of powerful regularization strategies of dropout, use of a new rectified linear unit instead of traditional sigmoid and hyperbolic tangent in combination to dropouts, and availability of large labeled training dataset for supervised training with the help of softwares like Mechanical TURK, Flicker etc. that

are used by ImageNet Challenge organizers to provide a very large labeled dataset for object recognition [29].

AlexNet [12] as shown in Figure 2.6 is a modified version of *LENET5* network. Not only this network is deeper than the traditional one, but wider and more complex as well. They distributed their network on two separate *GPUs* in such a way that two maps of layer 2, 4 and 5 interact only with in same GPU, whereas layer 3 is connected to all maps in layer 2. There are eight abstract layers including input and output but actually first two convolution layers contains one additional local contrast normalization and overlapped max pooling layers. Whereas, after last convolution layer there is an overlapped max-pooling layer followed by the three fully connected layers. The output of first two fully connected layers are 2×2048 (2048 for each GPU). However, last output layer is a fully connected to produce 1000 neurons using softmax function that represents 1000 categories of ILSVRC 2010 dataset. All convolution and fully connected layers are activated with $ReLU$ as shown in Eq. 2.18.

$$y = max(0, x); \tag{2.18}$$

The network consists of about 60 million parameter having a 10-bit constraint on mapping from image to label. However, to reduce over fitting they adopt two strategies, i.e. data augmentation and dropout. In data augmentation, the training data is being increased by taking patches from input image. It results an increase in data by a factor of 2048. Further, the intensities of RGB channels of images are normalized through PCA; multiply images with those Eigen values from PCA and then subtract mean zero and standard deviation of 0.1. Dropout is a probabilistic technique where output of 50% are dropped and set as zero, having no role in forward and backward propagation. However, in testing they multiply each output by 0.5 to achieve a geometric mean of predictive distributions produced by exponentially many drop out networks. Dropout avoids over fitting but increases cycles by reducing the convergence speed.

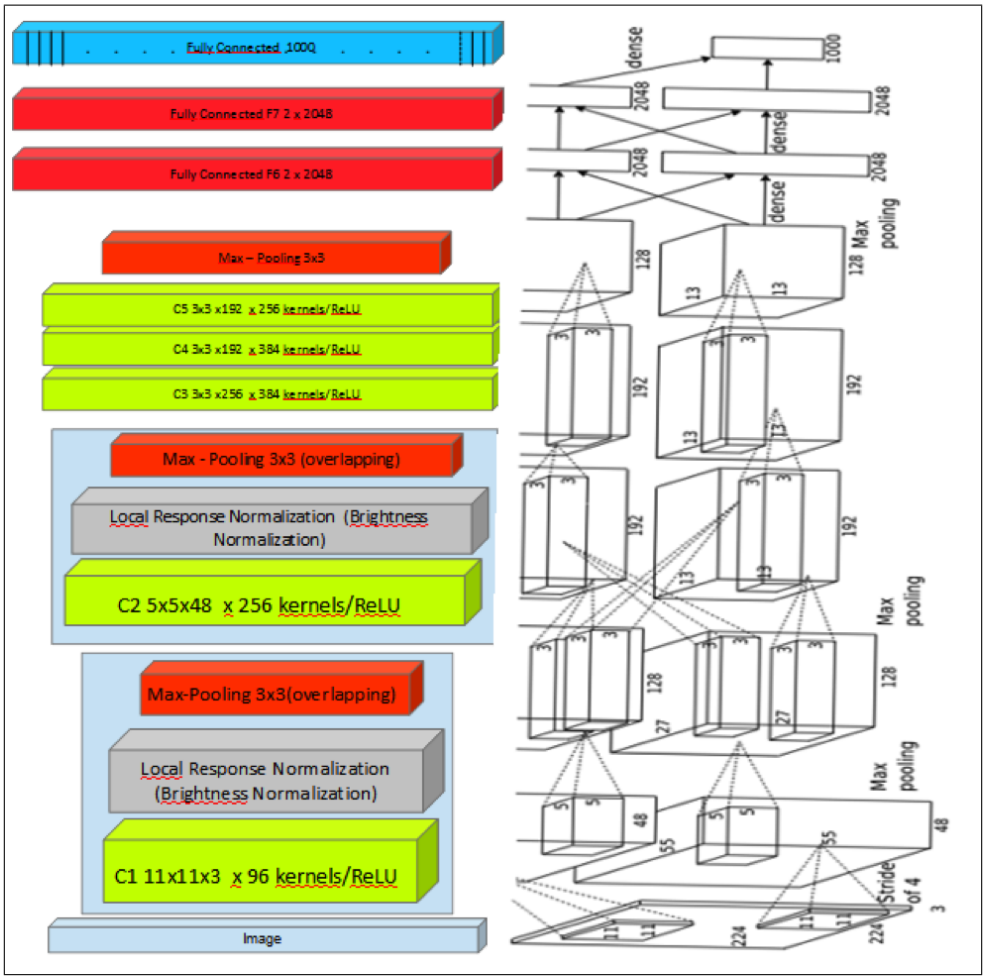


Figure 2.6: Alex ConvNet architecture on two GPU's, along with number of kernels and their sizes.

This model achieved error rate of about 17% that is more than 10% better than other state-of-the-art techniques such as sparse coding (similar to $BoVW$) with 28.2% and SIFT+FV with 25.7% in top-5 results on ILSVRC 2010. However, in top-1 difference is almost the same but the error is about 37.5%. Whereas, it achievs 15.3% error rate on ILSVRC 2012 testing set that can be seen in Tab. 2.1. This result is an abrupt jump of more than 10% decrease compare to the traditional $BoVW$ models that are competing since 2006 with reduction in points or ones from total error.

2.3.3 (RCNN)

One can sometime take benefit from cues such as prior knowledge of the input given to a network. In case of *OR* system, one of the cue can be detecting object as a target, that provide some benefits by localizing the object as a pre-processing step. Recently, R. Girshish et al. [92] propose a simple and scalable region based *CNN* detection algorithm *RCNN* that improve previous mean average precision (*mAP*) at about 30% compare to other reported maximum *mAPs* on VOC 2012 and ILSVRC 2013 detection dataset.

It mainly utilizes *CNN* to detect, localize and extract object patches from an image, and if the data is scarsed, then first supervised pre-training for auxiliary task is done. Followed by pruning the category specific training which helps distinctively in performance boost by utilizing low level cues such as color and super pixel consistency for object proposal in category-agnostic fashion. *CNN* classifier is used in the end to classify object categories at those locations.

Overfeat [93] is also similar to *RCNN*. It is another *CNN* based approach that work with multi-scale sliding window which could be considered a special case of *RCNN*, if one replaces selective search region proposal with multi-scale pyramid of regular square search region, change per class bounding box regressor to a single box regressor and does not use fine tuning for detection by *SVM*. However, OverFeat is 9x times faster than *RCNN* due to not being warped at sliding window at image level. This methodology is used and utilized by Oquab et al. [94, 95] for object recognition. The reason for discussing these object detection techniques is two fold: first it involved *CNN* and second they are used as a first step in *OR* that show very good results in recent papers.

2.3.4 Network-in-Network (NiN)

M. Lin et al. propose a novel deep model that uses a micro-MLP inside a traditional *CNN* that work like a network inside a network.

It is named 'Network-in-Network' *(NiN)* [96]. *BP* and multi layers of *MLP* are the similarity and motivation behind usage of *MLP* inside *CNN*, which they named as 'mlpconv' (a three layer perceptron) layer instead of general convolution model that does convolution and produce output after passing through activation function. In the end, instead they perform global average pooling instead of fully connected layers by producing a map for each class category of the last *mlpconv* layer that are averaged and fed directly to softmax layer. It provides three benefits (i) more native to convolution by enforcing correspondence between feature maps and categories (ii) over-fitting is avoided at this last layer because there is no parameter optimization and (iii) robust to spatial translation of the input due to adding spatial information.

To evaluate the performance of *NiN*, they use a stacked of three *mlpconv* layer followed by a spatial max pooling layer that does some reduction. Dropout is used in first two stacked layers to avoid over-fitting. They use the same setup for training and testing as done by *AlexNet* [12]: mini-batches of 128 samples, and used manually setup for learning parameters where as the learning rate is scaled down by 10 when the accuracy stops to increase. On CIFAR-10, the test error rate of 10.41% is recorded for model without data augmentation. However, when data augmentation using translating with horizontal flipping is used, it reduces it to 8.81% that overcomes all other stated state-of-the-art techniques with and without data augmentation. Similarly, on CIFAR-100 their model with dropout outperform other techniques by reducing 1.17% of error rate than others bringing overall to 35.68% without data augmentation. Their global average pooling provides a regularization to their model and reduced error rate from 17.56% to 15.99% on a similar network with and without global average pooling.

Zhouwen Tu et al. [97] propose a similar model, where they use infrastructure of Caffe [58], network architecture of *mlpconv* and global average pooling. However, to boost the recognition performance they

focus and emphasize on (i) transparency of middle level layers to overall classification (ii) robustness and discriminativeness of learned features (more importantly in starting layers) and (iii) avoiding the exploding and vanishing of gradients while training that reduces performance. Keeping these in mind, they propose a new deeply supervised net (*DSN*) algorithm in which they introduced 'companion objective' at each hidden layer that works as a regularizer similar to pre-training approach. *SVM* classifier is used at each layer for this companion objectiveness that behave like local supervision. In training, they adopted *SGD* approach similar as in *CNN*. However, in *BP* the error from output layer and the gradients also adds the local gradients from companion objectives at each layer.

A similar training setup like AlexNet [12] is used for *DSN* having momentum of 0.9, baseline infrastructure of *Caffe* [58], and network architecture of mlpconv with global average pooling [96]. This setup achieves 9.78% and 8.22% error rate on CIFAR-10 without and with data augmentation, respectively. It shows good results compared to others. Similarly, on CIFAR-100 they achieve stat-of-the-art results of 34.57% error rate. In addition, one positive impact is not using large datasets that reduces burden of having large training data. They show optimal results till date on MNIST, CIFAR-10, CIFAR-100 and SVHN datasets without data augmentation.

2.3.5 GoogleNet

Recently, Christian Szegedy et al. [66] propose the deepest *CNN* model which overcome all other state-of-the-art deep networks and other *CV* techniques in ILSVRC 2014 classification and detection challenge. *GoogleNet* with abstract 22 layered network is optimized with Hebbian principle, intuited by multi-scale processing, whereas, keeping computational budget constant despite of increasing depth and width such that it have 12x fewer parameters than *AlexNet* [12] and having more accuracy than previous models. This model has influence from *NiN* model [96, 66] and 'we need to go deeper' Internet

meme [98].

Two difficulties arises in increasing the depth, i.e. increase in computational resources and over-fitting. To tackle these issues, they suggest to move towards sparsely connected architectures from fully connected and even inside convolutions in Convolutional layer. The objective of Inception network is to find out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components to attain translation invariance by repeating the convolution. In the layers close to input image, correlated units use to gather in a local region that can result in number of clusters in same region which can be covered by convolution over 1×1. Whereas, there exists some small cluster far from each other that could be covered with larger convolutions of size 3×3 and 5×5. These three kernel sizes also avoids alignment issues and along with pooling of stride 2 represent the current architecture known as inception module that results in one bank of features as input to the higher layer. Output correlation statistics use to vary because inception modules are stacked over each other. Despite that the propose architecture may cover optimal sparse structure, it can lead to computational blow up with in few stages due to handling it inefficiently. To tackle this problem, they apply dimension reduction and projection where ever a threat arise due to increase in computational requirement by the help of 1×1 convolution before 3×3 and 5×5 and usage of $ReLU$ activation functions. In addition, this provide a multi-scale features to higher levels at the same time. The model is shown in Fig. 2.7 that won ILSVRC 2014 competition and 2-3x faster than similar performing networks with non-inception architecture.

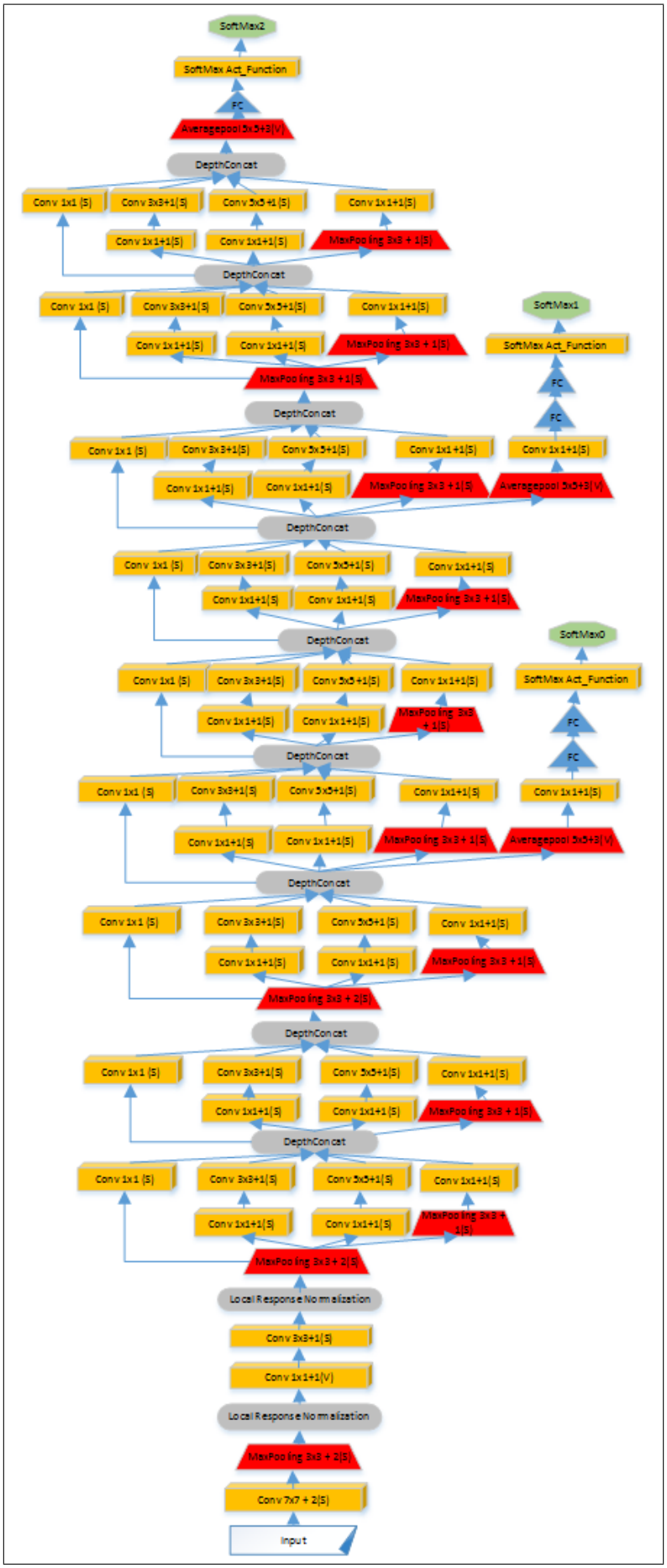


Figure 2.7: GoogleNet Model

In evaluation, by moving from fully connected layer to average pooling in presence of dropout, their top-1 accuracy increased by 0.6%. In addition, one important step in their network is introduction of auxiliary classifiers that are connected to intermediate layers in order to encourage discrimination in the lower stages by providing extra regularization. For training their *GoogleNet*, they adopt asynchronous *SGD* with momentum in addition to using Polyak averaging at inference time. It achieved top-5 error rate of 6.67% over ILSVRC 2014 without using any external data that outperformed all other.

2.3.6 3DCNN

3DCNN for *AR* from videos can be called as a pioneering role model in deep architectures [13]. It works end-to-end in a supervised manner for learning action classifiers. Their 3D volume learn implicitly spatiotemporal features. To effectively incorporate the motion information in video analysis, a 3D convolution is applied, which discriminate features along both spatial and temporal dimensions. The difference between the 2D and 3D convolutional kernel is shown in Fig. 2.8. In a 2D kernel, there is 1-1 communication between the input map and output map which learns spatial information (Fig. 2.8 (a)). Whereas, in a 3D kernel there is 3-1 communication as shown in Fig. 2.8 (b). The 2D kernel and 3D kernel equations are given below in Eq. 2.19 and 2.20.

$$v_{ij}^{xy} = tanh\left(b_{ij}\sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)}\right) \tag{2.19}$$

$$v_{ij}^{xyz} = tanh\left(b_{ij}\sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right) \tag{2.20}$$

Three weight kernels are applied on three input frames/maps in temporal domain that combines and generate an output map resulting

Table 2.1: Performance of DL for Object classification and recognition since 2010, Here Dropout, Sliding Window, Data Augmentation are represented by DO, SL and DA respectively. SetA represents ImageNet 2009 Fall version, SetB represents ImageNet 2012 + 22k Additional images, SetC Training set consist of PascalVOC (1000) + ImageNet (512) classes and Testing set consist of PascalVOC 2012 & 20 (but trained with 1512). While last SetD Training set contains PascalVOC (1000) + ImageNet (512) classes and for Testing (PascalVOC 2012) & 20 (but trained with 1512)

Model	Year	Dataset	Classes	Top-1	Top-5	Errate
TCNN [99]	2010	CIFAR-10	10	X	X	26.9%
AlexNet [12]	2012	ILSVRC 2010	1000	37.5%	17%	X
AlexNet [12]	2012	ILSVRC 2012	1000	36.7%	15.3%	X
AlexNet [12]	2012	SetA	1000	67.4%	40.9%	X
Clarifai [1]	2013	ImageNet 2012	1000	X	11.7%	X
Clarifai [1]	2013	SetB	1000	X	11.2%	X
NIN + DO [96]	2014	CIFAR-10	10	X	X	10.41%
NIN + DO + DA [96]	2014	CIFAR-10	10	X	X	8.81%
NIN + DO [96]	2014	CIFAR-100	100	X	X	35.68%
DSN + DO + DA [97]	2014	CIFAR-10	10	X	X	8.22%
DSN + DO [97]	2014	CIFAR-10	10	X	X	9.78%
DSN + DO [97]	2014	CIFAR-100	100	X	X	34.57%
GoogleNet [66]	2014	ILSVRC 2014	1000	X	6.67%	X
GoogleNet [66]	2014	ILSVRC 2014	1000	X	6.67%	X
Baidu [100]	2014	ILSVRC 2014	1000	X	6.67%	X

in spatial as well as motion information. In this model, first hard-coded layer (H1) gets 9 consecutive maps by leaving one out. H1 layer extract opical-flow along x-axis, optical-flow along y-axis, gradients along x-axis, gradient along y-axis, and original Grey level images, i.e. a total of 43 input maps of size 80×60 is given as input to the first 3D-convolutional layer as shown in Fig. 2.9. In addition, they apply multiple sets of distinct convolutional kernels at the same location on the input which extracts multiple types of features from the same location. 3D kernel not only reduces spatial dimensionality

of the maps with higher strides but also the number of features from three to one. That's why two sets of 3D kernels are used in order to increase information at higher layer of the model. This C2 results in 33×2 maps of size 72×54. Followed by a max-pooling layer that reduces the dimensionality of each map by 2x. Again a three sets of 3D kernels are applied on maps from pooling layer that provide 23×6 maps of size 18×12. Followed by a pooling layer that results in a 6×4 size map, and a *FC* convolutional layer that form a feature vector of size $128 \times 1 \times 1$. These features are used for classification of the action.

This model is applied on TrecVID a real world airport surveillance scenario that have recoding for several days. In addition, they apply it on KTH dataset. On TrecVID, it outperforms all others state-of-the-art techniques in three categories, i.e. Cell to ear, object put, Pointing Hand. However, on a small dataset, i.e. KTH compare to TrecVid, it has comparable results to the state-of-the-art techniques, i.e. 90.2% vs 91.7% for the six categories in KTH.

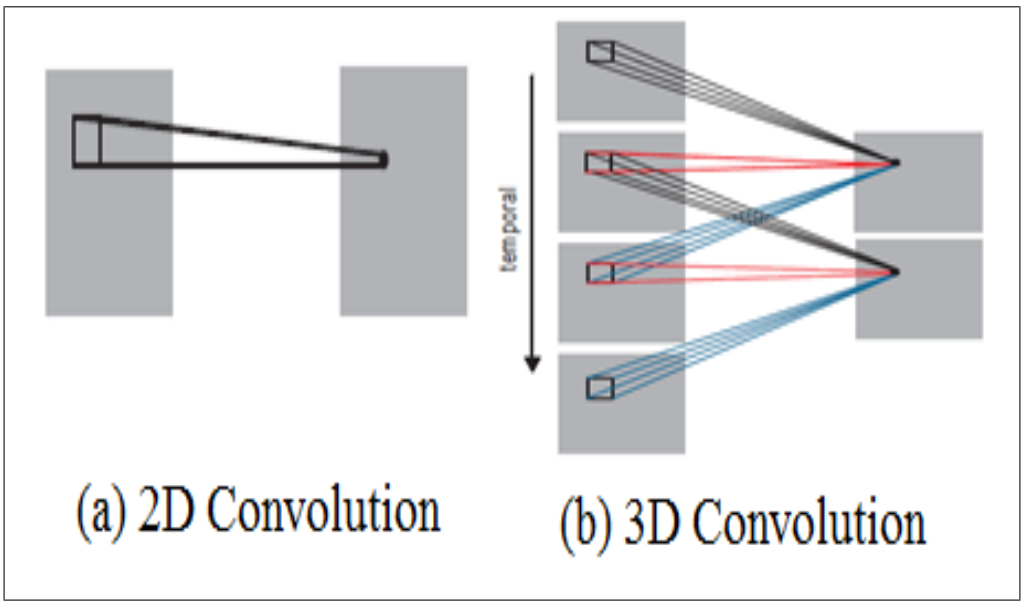


Figure 2.8: Difference between 2D and 3D Convolution

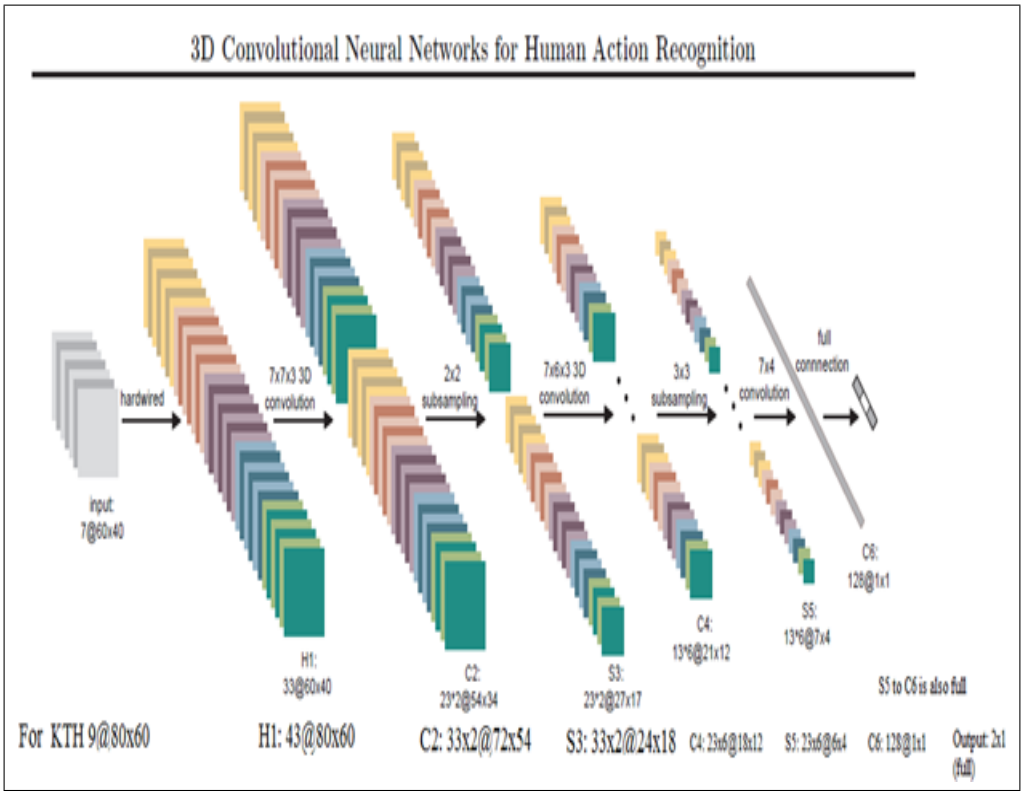


Figure 2.9: 3DCNN Architecture for Human Action Recognition

The hardwired layer and *CNN* based detected and extracted human ROIs make its functionality limited. In addition, its performance for small scale datasets is also low compare to other models. This shows that its limitation is mainly data and proper extracted and centralized humans as input to the network.

Further like other CNNs, it does not follow strict biological pyramidal structure of the brain. So we need new Bayesian approach for the weight kernels that works the same way but may provide better results. In addition, we can follow the strict biological pyramidal structure of the brain. Or may introduce some new weight matrix structure which may be able to absorb more information and may provide better performance even in the absence of huge amount of data and without using *CNN* as human detector, e.g. simple preprocessing as silhouettes [10, 101] or completely Grey level images as

input to the network.

2.3.7 Two Stream CNN

This model decomposes spatial and temporal components and deal them separately in advance as shown in Fig. 2.10. First module is called Spatial stream ConvNet that treat still frames as input to a deep ConvNet. While the second is called temporal stream ConvNet that takes motion information, i.e. optical flow between consecutive frames. The softmax output of both modules is lately fused and their score is calculated based on averaging or giving it to a multiclass linear SVM to find the classified category for the input.

The architecture of both the stream is similar, which contains 5 Conv layers with small 3×3 kernels and 3 fully connected layers. This architecture is being inspired by the model which won 2013 ILSVRC challenge with 13.1% top-5 error rate. The input image is of size 224×224. However, the spatial stream performs like image classification task for actions. They use a pre-trained model that is trained with ILSVRC-12 1.2 million images. And only retrained the last fully connected layer that is called classification layer for these video frames. This helps in saving a lot of time. Also literature shows that training a model with a pre-trained model has better performance. It take 25 frames from each video. Each frame is given to the ConvNet that becomes 10 by cropping from different positions and flipped images of the same image. Their results are averaged in the end. Whereas, the temporal stream takes dense optical flow information, i.e. displacement vector fields between pairs of consecutive frames as image that explicitly describe motion in videos and helps in making recognition easier as well as shows better results despite less training data. They have shown state-of-the-art results for UCF-101 and comparable results for HMDB-51 datasets.

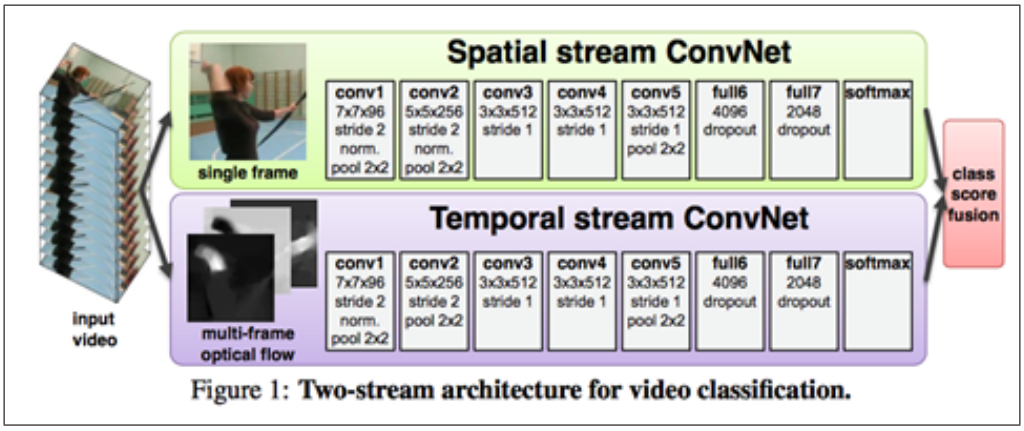


Figure 2.10: Two Stream model architecture for video classification

2.3.8 Learning Spatio-Temporal Features with Convolutional networks

D. Tran et al. [23] proposed a model that uses 3D *ConvNet* to learn spatio-temporal features. This 3D *ConvNet* is different than *3DCNN* [13] in following factors, 1) it uses full frame as input rather than ROI, 2) 3D pooling is used after 3D convolution rather than 2D pooling, 3) the temporal depth can be more than 3 (however, their experiments suggest 3 as the best temporal depth) 4) large input frame size, i.e. $16 \times 3 \times 112 \times 112$. However, the results are shown as binary class classification. In addition, it is pre-trained with Sports 1Million dataset, where the trained model is mainly used as a feature extractor. Those features are given to a linear *SVM* to classify as one-vs-all classification criteria. The features extracted are called *C3D* features.

The proposed model is a generalized model. They have evaluated and trained their model on Sports 1 Million dataset. Whereas, it extracts features for 4 different classification categories, i.e. action recognition (UCF101), scene recognition (YUPENN and MaryLand), action similarity labeling (ASLAN), and object recognition (ego centric object recognition). In all the scenarios, it outperformed all other recent models such as traditional *CV* handcrafted techniques [67], two stream [63], and A. Karpathy et al. DeepVideo [24].

The two-stream model [63] has different type of input spatiotemporal data, i.e. single frame to one stream and optical flow information to the second stream. These two stream are fused after softmax layer by calculating their class score. Based on their experiments, use of optical flow shows significant improvement over raw input frames despite small training data. J. Donahue et al. propose a model [64] similar to model in [60]. This long term recurrent convolutional (*LTRC*) network uses a *CNN* learned feature vector as input to a *LSTM*. However, this model is different in two ways, i.e. i) it integrate 2D *CNNs* rather than 3D that can be pre-trained on large datasets, ii) *CNN* and *LSTM* are combined into a single model to enable end-to-end fine-tuning. An additional difference can be the variable length input frames to the network. This model shows promising results for action/activity recognition.

J. Yue-Hei Ng et al. [65] propose a similar model, however they followed [63] model as the base structure to extract *CNN* feature vector. This feature vector is given to a feature aggregation module that perform different type of pooling or apply *LSTM* to predict the class predictions for each stream. In the end, fusion of class scores is being done. This model uses two *CNN* architectures, i.e. AlexNet [12] and GoogleNet [66].

2.4 Comparing CNN and PyraNet

In this section, we try to visualize, evaluate and understand either graphically or by explanation each layer of the *CNN* and *PyraNet* networks. More explanation is given regarding *CNN* compare to *PyraNet* due to the reason that there is a lot of work being done on *CNN* based models compare to others.

2.4.1 Visualizing and Understanding CNN

In this section, we have tried to answered some of the most asked questions by *CV* community. These questions can answer the old assumption against *NNs*, i.e. of being a BlackBox. Some of these questions are like:

- Whether *CNN* use background context or information or just the localize object for recognition?
- What each layer learn?
- Is there any sort of establishment or correspondence between specific objects parts to understand like human?
- Finding generalization power of deep *CNN* and effect of the depth and width?
- Will less training data affect overall performance?
- Does longer pre-training overfits the model generalization capability?
- Are *CNN* features sparse and does it affect?
- Finding grandmother cells which fire only for a specific classes that can help in increasing discrimination?
- Does spatial location matter for classification in *CNN*?

DL, especially *CNN* and its variants from last couple of years outperformed all other techniques, without any doubt. But still researchers can't fully understand the real recipe behind the success. Researchers are asking questions like previously mentioned, for which many are trying to answer by either visualizing the networks or through theory. However, most easy and affecting methodology is to visualize and understand. In this case some of the recent work done is in [66, 102, 1, 103, 104].

M.D. Zeiler and R. Fergus [1] answer few questions that why and how they work well and how can we improve them further. They visualize internal story of *CNN* as shown in Fig. 2.11. They used *Lenet5* and *AlexNet* [12] model and tried to reuse the process by recreating and going from output to input layer, i.e. instead of convolution, ReLU, and pooling, they did un-pooling, relu non-linearity and deconvnet using a transpose version of kernels with those rectified maps. From their experiments, they found visually that:

- Layer 2 captures information about corners, edges/color conjunctions
- Layer 3 captures fine grained complex invariances, e.g. textures
- Layer 4 captures class specific, e.g. faces legs
- Layer 5 shows entire objects with variation in pose e.g. dogs, keyboards etc.

Further to understand training and features relationship, the work in [1] revealed that lower layers compare to higher complex layers learn 10 times faster. In addition, [1] concludes that the characteristic of features are invariant to translation, scaling and rotational symmetry (only). Visualization of features highlighted certain limitations, e.g. first layer filters are mixture of high and low frequency information but does not care about mid-level. Similarly, at 2^{nd} layer a high stride is used with convolution that is losing valuable information. To fix these limitations, they reduce the kernel and stride

size due to which information is retained and it helps in classification improvement. Moreover, in another experiment in [1], objects are ocluded to answer the question whether *CNN* use background context, information or just the localize object for recognition. It answers in some probability of performance degradation. They also investigate and show that higher layer-5 do establish some form of correspondence between specific object parts, i.e. eyes and nose in faces. In the end, they try to find generalization power of deep *CNN* and effect of the depth and width. Last fully connected layer alone have no great effect on overall result, same is the case for last 2 convolution layers. But if fully connected in combination to middle 2 convolution layers are removed, performance falls that shows that depth does mater along with width of each layer. It is highlighted that if both width, depth and size of last two fully connected layers are increased, than it overfits.

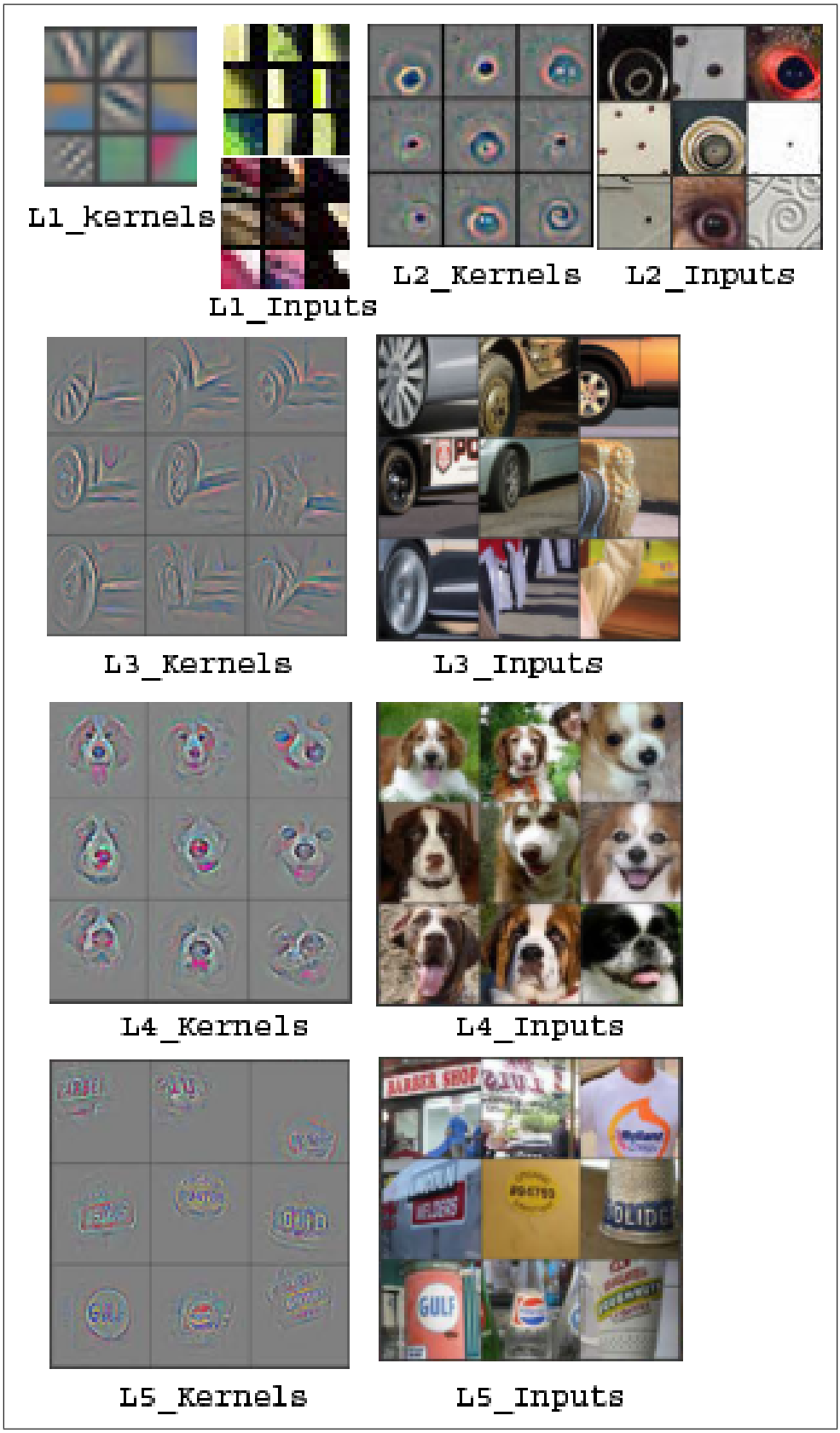


Figure 2.11: Visualizing feature maps of a deep CNN model [1]

The generalization power is shown using training a network using state-of-the-art ImageNet features for classification using a softmax function on Caltech and PASCAL VOC 2012 datasets. Only fully connected layers are fine tuned over old trained network of ImageNet. For Caltech-101, it beat state-of-the-art by 2.2%. Whereas, on Caltech-256 it shows huge difference, i.e. 74.2% vs. 55.2%. However, on PASCALVOC 2012 they are 3.2% lower than state-of-the-art. The reason could be that ImageNet and PASCAL VOC are quite different from each other. In the end, they further evaluate the importance of depth by using features from each layer independently. It is concluded that as the layer increases, the performance increases. Details list of results can be seen in their paper.

Similarly Agarwal et al. [104] try to answer some additional question with visualizing and evaluating same *AlexNet* model in certain extra scenarios, e.g.:

- Will less training data affect overall performance?
- Does longer pre-training overfits the model generalization capability?
- Finding grandmother cells which fire only for a specific classes that can help in increasing discrimination?
- Does spatial location matter for classification in CNN?
- Are CNN features Sparse and does it affect?

They perform similar steps as in [92] model, i.e. pre-training with ImageNet and fine-tuning on SUN (397 classes with 108k images) dataset. Whereas, testing it with PASCAL VOC 2007 (20 classes with 10k images). This gives optimal results of 52.2 ± 0.1 compare to simple fine tuning and training from scratch. Further, they visualize the fine tune result based on a measure of class selectivity by class label entropy of set of filters and thresholding those entropies. They report that fine-tuning changes entropy at layer 6 and 7 and these

entropies might be tuned with pre-trained model when data is limited. Otherwise, fine tuning the whole network may benefit more as well. They also found that generalization takes place quickly starting from layer 1 to 7. However, more pre-training does improve little bit of overall performance.

According to neuroscientists, Grandmother cells (GMC) are cells in human brain that fire only for specific visual saliency or stimuli, e.g. face involved in an object. According to Agarwal et al. [104] the *GMC* filter are those which have high average precision for classifying a class among other classes. However, it is revealed that GMC like filters exist only for few object categories like person, bicycles, etc. and compare to these classes, other classes require more filters to recognize an object class. It indicates that intermediate representations in *CNN* are distributed. Sparsity is considered an important point for different applications in *CV* and *ML*. They found that fully connected layers are mostly sparse. In the end, importance of spatial information is checked in each layer for overall performance and reported that in classification spatial information matters more at lower layer, i.e. layer 1, while gradually decrease is recorded at higher layers.

Till now researchers have found till layer 5 by visualizing and analyzing filters and maps to understand what each layer shows. But the question comes that if it learned till full body parts, what else can it learn after that. Can it narrate the story? Or can it enhance it by re-learning the features. Or does we need more layers to learn more general datasets compare to some specific datasets. We need to visualize networks like GoogleNet to further understand the learning of *CNN*.

2.4.2 Visualizing Pyramidal Neural Network

PNN or *PyraNet* are quite similar to *CNN*, e.g. same receptive field concept, activation functions, and several hidden layers. Still, there is very less work being done on prior networks. There are certain

main differences in weights and filtering methodology that differentiate both from each other. *PyraNet* first layer works similar to *BoVW* and *CNN* by searching and extracting low-level features from the image. However, unlike *CNN*, here the image is not as smoothed or edged as in *CNN*. *PyraNet* does down-sampling and feature extraction at the same time by reducing an image by:
$R_l = \lfloor (R_{(l-1)} - O_l)/G_l \rfloor$
$C_l = \lfloor (C_{(l-1)} - O_l)/G_l \rfloor$
where R_l and C_l are the number of rows and columns of the new map, respectively. $'l'$ represents the higher layer. O_l and G_l are the overlap and gap between two receptive fields at the layer. There is no separate layer for down sampling. High-level features are extracted continuously by repeating the same procedure until the size of the feature map is reduced to a specific size. The feature maps are too sparse due to the weighted sum operations as compared to *CNN*. Some researchers have tried some variation in changing the initial layers to hardcoded layer of Laplacian and Gaussian to provide filtered images. Also, contrast normalization has been utilized after each map. Finally, all the features are classified with a fully connected layer similar as a *CNN* model.

2.4.3 Similarities and Comparison

As previously discussed, *PyraNet* and *CNN* both have many things in common. Specially, the pyramidal structure and learning from photometric discontinuities. Here, we discuss the comparison of each layer of these models and tries to highlight some areas that can help us in improving efficiency of recognition processes. Starting from the input data, both use the same data to classify as class categories (for example Fig. 2.12 (a)). However, this input can be preprocessed in different categories for fast convergence or increase in recognition. Then comes the feature extraction stage, where *CNN* directly use convolution rather mainly cross-correlation for extracting feature from the receptive field and produces a smoothed edged image from

the beginning prior to learning the weights as shown in Fig. 2.12(c), *PyraNet* uses same convolution but with individual kernel for each receptive field due to unique weights of each neuron. It results in a sparse initial map shown in Fig. 2.12(b). Initially, it results in a sparse map, but creates edges after learning the weights.

Weights parameters are in various number and with specific architecture for each model. *CNN* has a kernel that is shared throughout a map, whereas *PyraNet* has a unique weight for each neuron at lower layer map which create a unique kernel for each receptive field that generates an output neuron. Hence, acquiring both properties from fully connected network and weight sharing due to having less fixed number of weights, similar to *CNN*. This big matrix of weights, and unique kernel of each neuron absorbs more features compare to one kernel of *CNN*. This is based on the ideology of strictly following biological phenomena and not caring about curse of dimensionality compare to *CNN*.

PyraNet maintains pyramidal structure by continuously reducing spatial resolution. It results in localizing the object and decrease in convergence speed and efficiency. Whereas, *CNN* decreases the map size by subsampling but increase its depth by using multiple kernels that reduce burden on shared kernels and learn different type of features with different kernels. However, this increase of kernels result in violation of pyramidal architecture, increase in time complexity and convergence time. *CNN* shows that if it does not increase the depth, performance reduces and if increased more than a specific limit, it over-trains [1]. In addition, invariance to geometric transformations of the input are achieved with continuous reduction of spatial resolution while increasing continuously the feature maps in each higher convolution layer that provides more richness of the representation in *CNN* compare to *PyraNet* due to its less variant maps in higher layers.

In terms of model structure, *PyraNet* has almost same architecture, i.e. Normalization and weighted sum that is repeated until

last fully connected layers that does the classification similar to the *CNN* last fully connected layers. *CNN* on the other hand is changing continuously, e.g. Normalization, Convolution and than pooling, to Convolution, Pooling and than consecutive convolution layers. Or even like recent new GoogleNet model, that have a softmax classifier at end of pooling in each layer to enhance the low level features and in order to generate better optimal result by not vanishing the gradients [66].

PyraNet and *CNN* both requires large training data to learn properly. In *CNN*, invariance to geometric transformations of the input is achieved with continuous reduction of spatial resolution while increasing continuously the feature maps in each higher convolution layer that provides richness of the representation. However, one question arises that whether the difference of performance between *CNN* and *PyraNet* is the weight parameters or the absence of down-sampling (Pool) layer that provide invariance to geometric transformation of the input. Will *CNN* give similar performance if we remove down sampling layer but keep weight sharing concept?

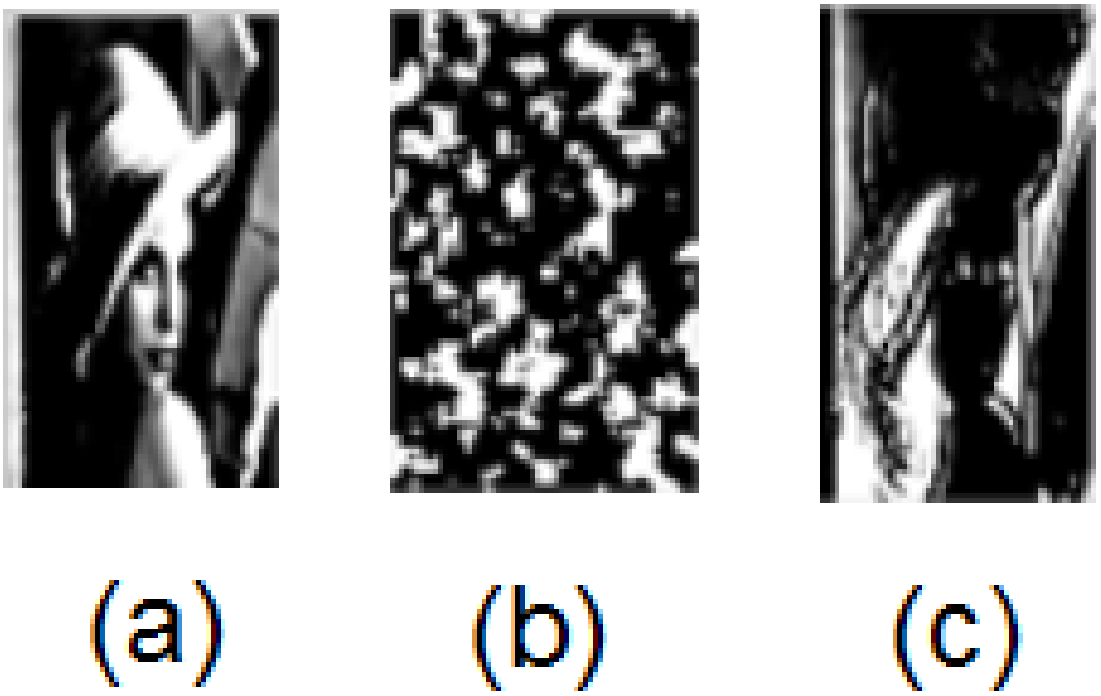


Figure 2.12: Comparison of Convolutional kernel vs Weighted Sum Kernels, (a) Original Leena Image (b) Output map resulted by Weighted Sum Kernels of *PyraNet*, (c) Output map resulted by Convolutional kernel of *CNN*.

2.5 Chapter Summary

This chapter helps in reviewing how the performance of *DL* techniques enhanced. Even now question arises such as whether this performance is due to huge amount of label data or latest powerful computational power. One need to understand the structure of these techniques. In order to benefit from techniques like *PNN*, which are similar to *CNN* and follow the same visual cortex hierarchy of brain, but with little difference in weight architecture.

Reviewing previous work being done on *PNN* and *CNN* model, considering their benefits and limitations, for example, parameters explosion is one of the problem that need to be tackled. Further, rather than the same weighting scheme of *CNN*, we decided to propose a new deep model with new weighting scheme. Therefore, we

have concluded that a pyramidal structure and the weighting scheme being used in *PyraNet* can be a viable solution. Not only for parameter reduction, but also to enhance the performance of various applications. We can introduce a new model by combining the good points of *3DCNN*, *C3D*, *PyraNet*, and pyramidal structure. In the next chapter, our proposed 3D pyramidal neural network is explained.

Chapter 3

A 3D Strictly Pyramidal Neural Network

Our Contribution:

- Presented a novel biologically inspired 3D pyramidal neural network architecture for multi-class classification.
- Extension of *PyraNet* 2D weighting scheme to 3D weighting Scheme.
- Introducing 3D temporal pooling.
- Modifying back-propagation for training the *3DPyraNet*.
- Evaluating performance on *AR* and *DSR* benchmark datasets.

3.1 Motivation

The traditional *NN* accepts an input or a vector, and simplify it through a sequence of hidden layers. Each hidden layer consists of multiple neurons, where each neuron is fully connected to all neurons in lower layer. Further, each neuron in a single layer functions completely independent and do not share any connections, i.e. no weight sharing. The last fully-connected layer is called the "output layer" and in classification problem, it signifies the target class scores. A neuron in hidden layer with this network structure having input image size of $32 \times 32 \times 3$ (cifar-10 image) consists of 3072

learnable parameters. However, if there are several hidden neurons, then multiply it with those numbers of neurons. This results in enormous learnable parameters that are not only hard to manage but also results in over fitting.

The biological receptive field concept solves this problem as we studied in previous Chapter. 2. Y. Lecun famous convolutional kernel reduced these parameters enormously. However, still it have some limitations. Learning huge data in today's large amount of data, puts burden on these fewer number of parameters due to which current models increase kernels as the network goes deeper. Therefore, we need a solution that combines both these approaches without disturbing accuracy and increasing the learnable parameters.

3.2 3DPyraNet

We report a strict 3D pyramidal neural network *(3DPyraNet)* model. It is based on *CNN* weight sharing concept and the *IP* structure. *3DPyraNet* extracts features from both spatial and temporal dimensions by keeping biological structure, thereby it is capable to capture the motion information encoded in multiple adjacent frames. One outlined advantage of *3DPyraNet* is that it maintains spatial topology of the input image by having position oriented parameters and presents a simple connection scheme with lower computational and memory costs compared to other neural networks.

3DPyraNet deep architecture is designed for a general scenario from videos, e.g. *AR* or *SR*, by taking inspiration from *3DCNN*, *PyraNet*, and recent *C3D* discussed in Sec. 2.3 and 2.2.5, respectively. This strict 3D pyramidal architecture is based on decision making pyramidal structure of a brain and pyramidal neurons. This approach is widely used in *NN* models, as discussed in previous chapter.

In *3DPyraNet*, we proposed three main modules, i.e. 3D weight matrix for 3D weighted sum/correlation operation, 3D temporal pool-

ing, and most importantly the pyramidal structure. We discuss all of these one by one in coming sections. Similar to *3DCNN*, *3DPyraNet* start with a big input data stream followed by extraction of different set of feature maps with randomly initialized several sets of weight matrices in first layer. These feature maps are refined at each higher layer until we achieve a reduced most discriminative set of feature vector for classification of action/scenes in the videos. Our aim is to reduce ambiguity in those extracted features and eventually, to enhance the performance. *IP* techniques use similar approach of extracting fewer features through coarse to fine refinement. In addition, the structure of image pyramids and *NN* are also similar. For this reason, *3DPyraNet* model is developed by taking inspiration from an early strictly pyramidal *NN* model [11] and image pyramids approach.

To capture actions/scenes as a whole from the videos, we adopt a similar weighting scheme as used in *PyraNet* (previously discussed in Sec. 2.2.5.2). To take advantage of temporal information in the videos, we modify the 2D structure (Sec. 2.2.5.2) to 3D, and adopted the 3D structure (sec. 3.2.1). This modification to 3D is done by taking inspiration from *3DCNN* model [13]. The parameters are learned from input till output using a modified structure of traditional back-propagation algorithm.

The objective is to show that strictly following pyramidal structure and adopting a pyramidal 3D structure can enhance performance by learning spatio-temporal features compare to unrestricted models even with simple structure, fewer feature maps and hidden layers.

3.2.1 3D Weight Matrix

The concept of 2D weight matrix introduced in *PyraNet* is modified for 3D structure by using three weight matrices at a time as show in Fig. 3.1. This helps in incorporating the temporal information from the given input consecutive frames. In order to capture further various type of features, several sets of weight matrices are used.

We randomly initialize these weights on each layer but taking care of suggested techniques stated in literature for corresponding activation functions used at those layers. For example converting all the weights to have zero mean and unit variance.

This 3D weight matrix approach is not similar to 3D kernels of *3DCNN*. They have a small 3D kernel that filters the whole frame or feature map and does convolution on three consecutive maps to produce one output map. Whereas, our approach have three dimensional weight matrix of the size of input map. A small empty kernel of size r (also represented as RF in some places of this work) is taken to calculate the weighted sum or correlation operation between adjacent neurons of the input frame/map and weight parameters of the weight matrix, just like a 3D kernel. However, each neuron gets a unique 3D kernel form the 3D weight matrix. This correlation function instead of convolution tries to search for correlation among the consecutive receptive fields of the frames to recognize specific category.

The weight-sharing is different than traditional one. The weight sharing is very minimal in this approach i.e. in worst case

$$\frac{r}{r \times r}$$

where $r = 3, 5, 7 \ldots$ but less than the matrix size. r is the size of the receptive field kernel. Each neuron has its weight parameter that is shared locally whenever that neuron is used in a receptive field of an output neuron. Each neuron in output map has a position oriented locally connected kernel in the weight matrix. This technique helps to reduce the ambiguity and burden on the weight kernels. As it learns position oriented features.

To visualize and examine what weight matrix has learned, we use a sub-model without pooling ($L2P$) layer, making it similar to *PyraNet* model but in *3DPyraNet* structure as shown in Fig. 3.5. The model is trained with batch mode on Weizmann dataset for about 500 epochs until it converged at 450^{th} epoch. Than we visualized the learned

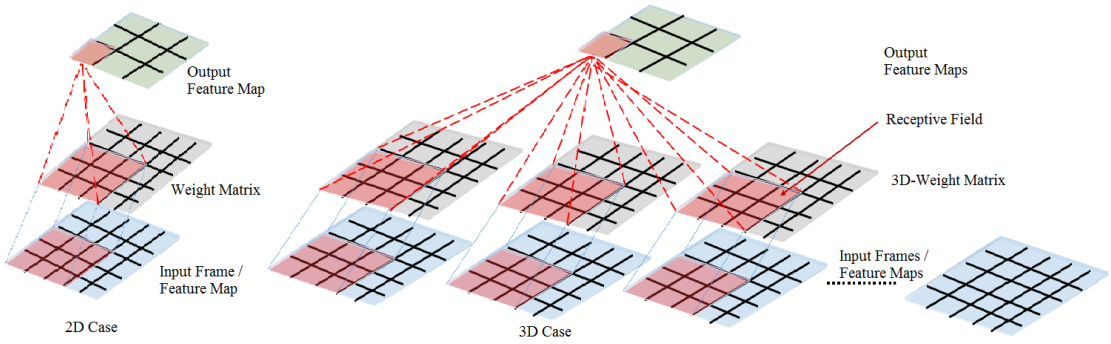


Figure 3.1: Difference between 2D and 3D weighted Sum Kernel Calculation

3D weight matrix parameters of first layer. We analyze that those three learned weight matrices are different from each other in-terms of texture, illumination, and the position of most activated neurons. Initially, feature maps produced by *WS* kernels are sparse as compare to convolutional kernel. But later with training the model, it becomes similar to smooth blurred images of the input sequences as shown in Fig. 3.3 and 3.4.

There are two main variables that effect performance of a network in this approach i.e. receptive field size and overlap. We have examined the performance with different sets of these parameters that are discussed in coming sections. However, from our experience, the most considerable receptive size is 3-5 with an overlap of 2-4.

3.2.2 Proposed Architecture

The basic *3DPyraNet* model has three main layers i.e. *3Dweighted-Sum or 3DCorrelation*, *3DPooling*, and fully connected *FC* layers. The first network consist of an input layer, three hidden layers, and a *FC* layer as shown in Fig. 3.5. Some recent papers show activation step as a separate layer as well as normalization as another separate layer, than based on that our model consist of ten layers. It makes it a deep network. Tab. 3.1 shows details about certain variable and parameters used in forward propagation phase.

The given input for this model is in silhouettes/binary form. No

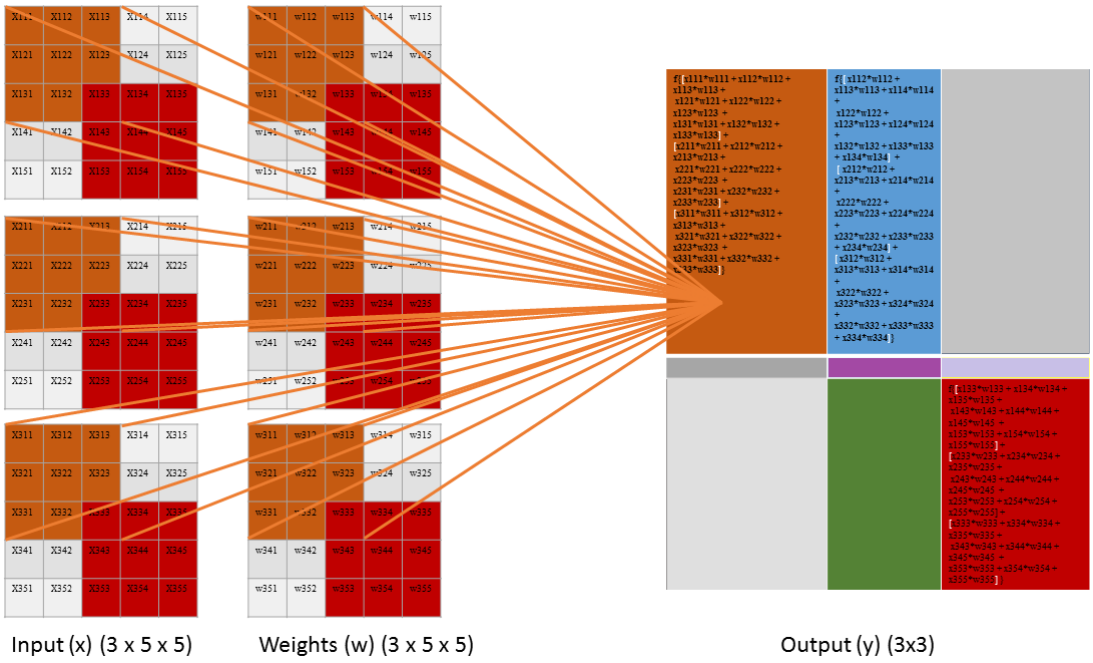


Figure 3.2: 3D weighted Sum Kernel Calculation

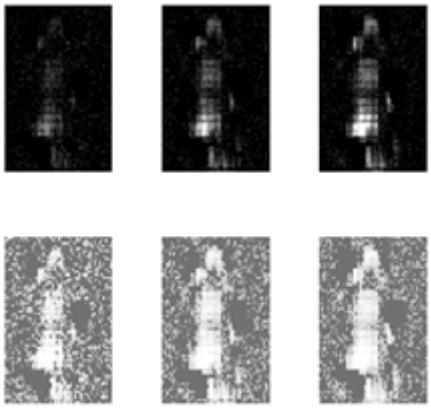


Figure 3.3: Learned 3D kernel Matrix of first layer of a trained 3DPyranet model

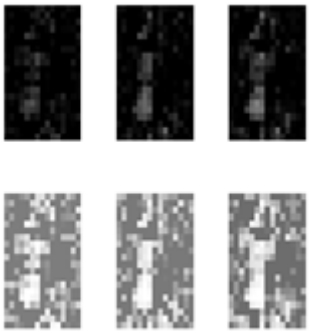


Figure 3.4: Learned 3D kernel Matrix of second layer of a trained 3DPyranet model

specific or sophisticated pre-processing is done as compared to existing *DL* model [13] for videos. The only pre-processing being done for *AR* is applying *SOBS* background subtraction algorithm [10]. Whereas, in case of dynamic scene recognition a full gray level frame is give as input to the network. However, the images are normalized between 0 and 1 by applying zero mean and unit variance procedure for both *AR* and *DSR*. It helps in faster convergence and better performance.

3.2.2.1 3D Correlation Layer

In general, temporal part gives correlation among the objects or actions in consecutive frames of a video. Therefore, first hidden layer is a 3D weighted sum (WS) or correlation layer represented as ($L1WS$) in the model (Fig. 3.5). It results in maps containing spatial as well as temporal information extracted from the given sequence of input frames at input layer. This (WS) layer is a pure correlation operation among the given neurons and weights in a receptive field of a frame and weight matrix as shown in Eq. 3.1. The output of 3D Correlation kernel is passed through an activation function i.e. hyperbolic tangent or LRelu. The resulting map is once again normalized before passing it as input to the next layer. This normalization not only enhance accuracy by $4-5\%$, but it also helps in faster convergence

Description	Notation	Details
Input Clip Size	N_1	$N_1 = H_1 \times W_1 \times M_1$
Number of Input frames/maps	M_{l_n}	$\begin{cases} if \;\; l_n = 1 \;\; than \;\; M = 13 \\ otherwise \;\; M = Z_{l_n} \end{cases}$
Number of Output feature Maps	Z_{l_n}	$Z_{l_n} = \left(M_{l_{n-1}} - D + 1\right)$
Type of Layers	*3DCorr* or *3DWS* *3DPool*, *FC*, *L*	3DCorrelation (weighted sum), 3DPooling, Fully Connected, and Total Layers
Last pyramidal layer	l^P	it is converted to 1D layer
Activation function of layer l_n	f_{l_n}	where $n = 1, 2, \ldots L$, *Sig, htan, ReLU, LReLU*
Size of Receptive field in pyramidal Layers l_n	RF or r_{l_n}	$l_n \neq l^{fc}$
Overlap in each receptive field of layer l_n	o_{l_n}	$l_n \neq l^{fc}$
Stride/Gap for pyramidal layer l_n	g_{l_n}	$g_{l_n} = r_{l_n} - o_{l_n}$
Size of feature map at layer l_n	$H_{l_n} \times W_{l_n}$	$H_{l_n} = \left\lfloor \frac{H_{l_{n-1}} - o_{l_n}}{g_{l_n}} \right\rfloor$, $\mathrm{W}_{l_n} = \left\lfloor \frac{W_{l_{n-1}} - o_{l_n}}{g_{l_n}} \right\rfloor$
Temporal Depth	D	D=3
Frame Stride	G	$G = 1$ each time we leave 1 frame
Weight parameter (i, j, d) at correlation layer	$w^{l_n}_{i,j,d}$	where $i = 1, 2..., H_{l_{n-1}}$, $j = 1, 2..., W_{l_{n-1}}$ and $1 \leq d \leq D$
Bias Parameter (u, v, z) at correlation layer	$b^{l_n}_{u,v,z}$	$u = 1, 2 \ldots H_{l_n}$, $v = 1, 2 \ldots W_{l_n}$ and $z = 1, 2, \ldots \left(M_{l_{n-1}-D+1}\right)$
Pooling Layer Weight parameter at (i,j)	$w^{l_n}_{i,j}$	where $i = 1, 2 \ldots, H_{l_{n-1}}$, $j = 1, 2 \ldots, W_{l_{n-1}}$ and $1 \leq d \leq D$
Pooling Layer Bias parameter at (i,j)	$b^{l_n}_{u,v,z}$	where $(u, v, z) = 1 \times 1 \times Z_{l_n}$
Total number of Parameters in the Model	P	$P = \sum_{l_n=1}^{l_p} N_{l_n} + \sum_{l_n=l_{p+1}}^{L} N_{l_n} \times (N_{l_{n-1}} + 1)$

Table 3.1: 3DPyraNet Mathematical Notations to be used in Forward Propagation

of the network.

$$y^{l_n}_{u,v,z} = f_{l_n}\left(\sum_{d=1}^{D} \sum_{(i,j,m) \in R^{l_n,d}_{(u,v,z)}} \left(\left(w^{l_n}_{(i,j,d)} \cdot y^{l_{n-1}}_{(i,j,m)}\right) + b^{l_n}_{(u,v,z)}\right)\right) \quad (3.1)$$

Where f_{l_n} represents activation function used at current layer l_n. (u, v) represents the position of output neuron at z, where z is the current output map that is generated by a set of input maps (repre-

sented by m) in the temporal direction. m is calculated by '$d+z-1$' from layer 'l_{n-1}' as shown in Eq. 3.2 third row. In our experiments, d_{low} and d_{high} are set equal to 1 and D, respectively, due to the size of the kernel temporal depth. The set of neurons of a receptive field, i.e. i, j in the current map m at the lower layer is calculated by Eq. 3.2. Where $R^{l_n,m}_{(u,v,z)}$ represents the receptive field for each neuron (u, v) in z output map. Here, r_{l_n} in the equation represents the size of the receptive field, i.e. RF at that layer l_n.

$$R^{l_n,d}_{(u,v,z)} = \left\{ \begin{array}{c} (i,j,m) \mid (u-1)+1 \leq i \leq (u-1)+r_{l_n}; \\ (v-1)+1 \leq j \leq (v-1)+r_{l_n}; \\ (d_{low}+z-1) \leq m \leq (d_{high}+z-1) \end{array} \right\} \tag{3.2}$$

In case of biases, unlike *CNN's*, *3DPyraNet* does not use one bias for each output feature map, rather it uses one bias for each neuron in an output feature map. Further, similar to *3DCNN*, we also tried to extract more type of features from the same spatial position of the input by using multiple 3D weight sets. However, *3DCNN* increase their set of kernels at each layer, whereas, *3DPyraNet* keep it fixed after starting layer. At each upper layer, the maps are decreased by two maps. Initialization with multiple kernels at first layer helps not only in increasing feature maps, but afterwords, due to continuous reduction in maps, help *3DPyraNet* in maintaining a pyramidal structure.

(RF) and (O) are tuned for handling the performance. *(RF,O)* size of (4,3) and (3,2) are used in *3DCORR1* and *3DCORR5* layer of the models for AR, respectively. Similarly, the model for DSR uses *(RF,O)* size of (4,3) in *3DCORR1*, and *3DCORR5* layer, respectively.

The reason behind using a correlation '.' operation in Eq. 3.1 rather than convolution '$*$' is that, correlation extracts and collect similarity. As action/scene is defined by recognition of consecutive almost similar activity, pose of a human body over a continuous time span, or continuous natural phenomena occurring over time in

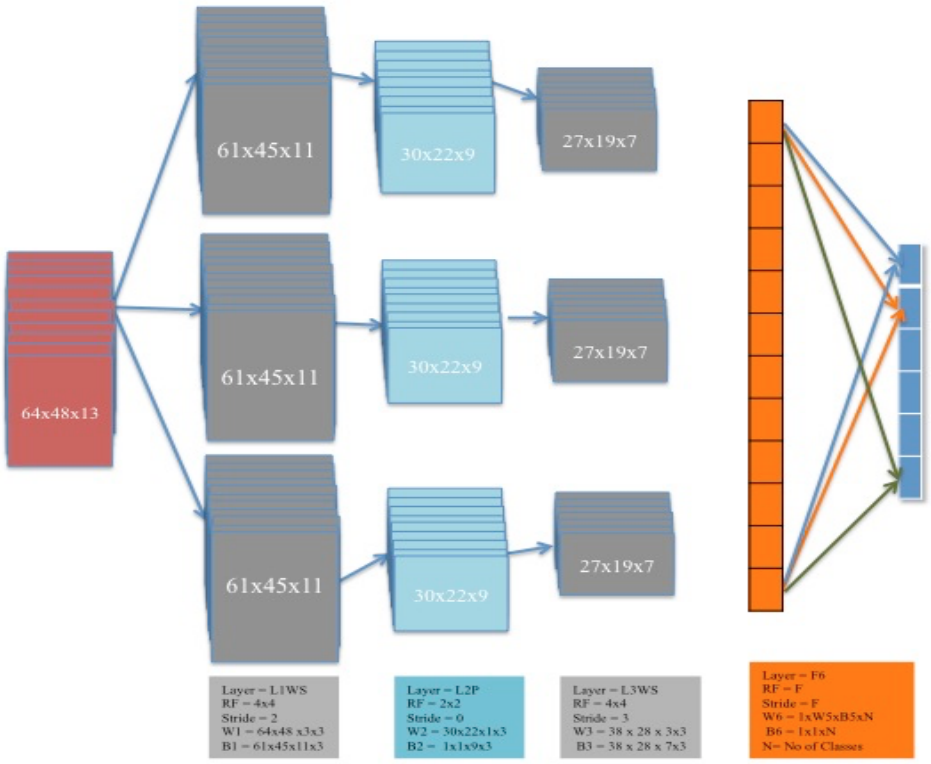


Figure 3.5: Proposed model of 3DPyraNet

a specific location. Therefore correlation or weighted sum operation is most suitable for recognizing similarity in videos due to the correlation existence in consecutive frames.

The WS layer has two main tunable parameters i.e. receptive field size and stride/overlap for handling the performance. We used three sets of 3D weight matrix in order to extract different type of features from the actual input. The set of weights remain same whereas their size reduce through out the network until 1D layer. In addition, maps decrease by two in each set as we go deeper in the network. In each layer, after passing the feature maps through activation function, the output maps are normalized in order to converge the network faster by regulating their saturation with simple zero mean and unit variance approach. This normalization steps also have an impact on overall performance. If we consider this step as a separate layer, than

our model becomes more deeper like some recent deeper model have shown. We tried to capture global as well as local discriminative features among the consecutive correlated feature maps. Therefore, $L1WS$ is followed by pooling layer that is discussed in next section.

3.2.2.2 Temporal Pooling Layer (3DPool)

Pooling divides the feature map into a set of non-overlapping rectangles. For each such sub-region it returns a value among each division. One can use different type of pooling, e.g. Mean, Max, etc. *3DPyraNet* uses Max-pooling as it returns maximum value among each division. This is helpful in removing non-maximum values that reduces computation for higher layers as well as provide translation invariance and robustness.

A $3D$ temporal pooling layer represented by ($L2P$) in our model is shown in the Fig. 3.5. This not only reduces spatial resolution but also, due to $3D$ pooling among the temporal domain, leads to more discriminative feature maps. The position oriented weight matrix has a slight deficiency of learning translation and scale invariant features. Therefore, to overcome this limitation, we introduce a pooling layer that helps in translation and scale invariant problem. In-addition, it also helps in reducing the dimensionality not only in spatial domain but also in temporal domain to maintain a strict pyramidal structure. One point to note, unlike traditional pooling layers, where there are no parameters or biases. Our model have weight parameters for each output maximum value among the three referenced fields. After multiplication, a bias is added and the signal is passed through an activation function like a traditional perceptron to produce an output map. The output max pooled value for neuron $y^{l_n}_{u,v,z}$ is shown in Eq. 3.3.

$$y^{l_n}_{u,v,z} = f_{l_n}\left(\left(w^{l_n}_{u,v} \cdot \max_{1\leq d\leq D}\left(\max_{(i,j,m)\in R^{l_{n-1},d}_{u,v,z}}\left(y^{l_{n-1}}_{i,j,m}\right)\right)\right) + b^{l_n}_{u,v,z}\right) \tag{3.3}$$

Here $R_{u,v,z}^{l_n,m}$ calculates the range for (i, j, m) indices, i.e. i_{low}, j_{low}, i_{high}, j_{high}, m_{low} and m_{high} as being calculated by Eq. 3.2 in previous layer. The temporal depth 'D' is same as other layers. However, the size of the receptive field (RF also represented as 'r_{l_n}' in Eq. 3.2) is different than the receptive field of *L1WS/3DCorr* layer. In our current models for AR and DSR, *(RF,O)* values are taken as (2,0) in first pooling layer. In pooling layer, we have taken weight matrix of size equal to the output map size, whereas, we set only one bias for each output map. Hence, u and v in Eq. 3.3 are 1, i.e. the bias matrix is $1 \times 1 \times Z$. where Z is the number of output maps in that layer.

3.2.2.3 Fully Connected Layer

The third layer is again a correlation layer (*L3WS*). Its output is converted in 1D column feature vector that is used as a fully connected layer for classification. The overall *L3WS* and *L2P* layers extract discriminative features by capturing the motion information encoded in multiple adjacent frames. One can have multiple 1D layers depending on the complexity of the application area. Eq. 3.5 calculates the output for another 1D layer or the final output that categories the input sequence.

In order to comply with other models, and for the ease of future reader, we have used the same neuronal structure for our final output layer to classify the features in output neurons as shown in Eq. 3.4. Otherwise, the same can be represented in Eq. 3.5, where $'u'$ and $'z'$ are one as we have $1 \times n$ output vector.

$$y_n^l = f_l \left(\sum_{m=1}^{N_{l-1}} \left(w_{m,n}^l \ . \ y_m^{l-1} \right) + b_n^l \right) \tag{3.4}$$

$$y_{u,v,z}^{l_n} = f_{l_n} \left(\sum_{i=1}^{I} \left(\left(w_{(i,v,z)}^{l_n} \cdot y_{(i,1,1)}^{l_{n-1}} \right) + b_{(1,v,1)}^{l_n} \right) \right) \tag{3.5}$$

However, in 1D case the u and z are 1. Only v is more than one, i.e. the number of output neurons (V). In addition, the set of weights for that is represented by v in the weight matrix. I represents the number of weight parameters or the number of neurons in the 1D current input layer. If $l_n = L$, than it is the final output layer otherwise, it is a 1D fully connected layer (FC) that is given as input to calculate output. Weight update is done using a modified version of conventional BP algorithm with a stochastic gradient decent approach for minimizing the error.

3.2.3 3DPyraNet Training

To learn *AR/DSR* task efficiently, a fast training algorithm must be devised. The objective of *3DPyraNet* training is to minimize gradually an error function that is defined in terms of our deep-network outputs and the target outputs. There are two well-known error functions used to minimize the error. However, literature review suggests that *CE* perform similar or better than *MSE*. The difference in computation comes only at final output layer. The rest of the computation is the same. Hence, we discuss the computation over *MSE*. Both these errors are explained earlier in *PyraNet* model Sec. 2.2.5.3. Delta rule given in Eq. 3.6 is used to update the weight parameters.

$$w_{u,v,d}^{l_n,new} = w_{u,v,d}^{l_n,old} - \varepsilon \frac{\partial E}{\partial w_{u,v,d}^{l_n}} \tag{3.6}$$

Where E is:

$$E = E_{MSE}(w) = \frac{1}{K \times N_L} \sum_{k=1}^{K} \sum_{n=1}^{N_L} |y_n^{L,k} - t_n^k|^2 \tag{3.7}$$

and ε is the learning rate that controls the oscillation while training. It is always kept small. We use a small value and reduce it by a factor of 10% after each 10 epochs. This remains the same for all unless it is otherwise specified.

Description	Notation	Details
Training Clip Index	k	$k = 1, 2 \dots K$
Target output category	$t^k_{u,v,z}$	$t^k_{u,v,z} = \left(t^k_1, t^k_2, t^k_3, \dots t^k_I\right)^T$ where $u = 1, z = 1$ and $v = 1, 2 \dots I$
Weighted Sum input to neuron (u,v,z) in Correlation Layer	$S^{l_n,k}_{u,v,z}$	$S^{l_n,k}_{u,v,z} = \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{m=m_{low}}^{m_{high}} \left(\left(w^{l_n}_{(i,j,d)} \cdot y^{l_{n-1}}_{(i,j,m)}\right) + b^{l_n}_{(u,v,z)}\right)$ where $i_{low} = (u-1)g_{l_n} + 1$; $i_{high} = (u-1)g_{l_n} + r_{l_n}$; $j_{low} = (v-1)g_{l_n} + 1$; $j_{high} = (v-1)g_{l_n} + r_{l_n}$; $m_{low} = d_{low} + z - 1$; $m_{high} = d_{high} + z - 1$;
Output Neuron at Correlation layer (l_n) for sample k	$y^{l_n,k}_{u,v,z}$	$y^{l_n,k}_{u,v,z} = f_{l_n}(S^{l_n,k}_{u,v,z})$
Softmax Mapping for Sample k	$p^{k,L}_{u,v,z}$	$p^{k,L}_{u,v,z} = exp(y^{k,L}_{(u,v,z)}) / \sum_{i=1}^{I} exp(y^{k,L}_{(u,v,z)})$
The error at output	$e^k_{u,v,z}$	$e^k_{u,v,z} = \begin{cases} y^L_{(u,v,z)} - t^k_{(u,v,z)} & \text{for MSE} \\ \mathrm{p}^L_{(u,v,z)} - t^k_{(u,v,z)} & \text{for CE} \end{cases}$
Error Functions	$E(w)$	$E_{mse} = \frac{1}{K \times N_L} \sum_{k=1}^{K} \sum_{n=1}^{N_L} \lvert y^{L,k}_n - t^k_n \rvert^2$ $\mathrm{E}_{ce} = \sum_{k=1}^{K} \sum_{n=1}^{N_L} t^{k,K}_n ln p^{k,L}_n$
Error Sensitivity of neuron (u, v, z)	$\delta^{l_n,k}_{u,v,z}$	$\delta^{l_n,k}_{u,v,z} = \frac{\partial E}{\partial S^{l_n,k}_{u,v,z}}$

Table 3.2: 3DPyraNet Mathematical Notations to be used in Backward-Propagation

In next section we explain the computation done for calculating $\frac{\partial E}{\partial w^{ln}_{u,v,z}}$. For *FC* layers, it is straight forward like a multilayer perceptron. However, in pyramidal layers it becomes complicated. We explain it separately for each layer, i.e. 3DPyramidal layer, 3DPooling Layer and fully connected layer.

Tab. 3.2 represents variable and notations being used in backpropagation stage. In order to update the weights, we have to calculate $\frac{\partial E}{\partial w^{ln}_{u,v,z}}$ for every layer. However, we divide it into two steps. The first step calculates error sensitivity or local error of each neuron, while in second step we calculate the weight gradients to be subtracted from previous weights.

3.2.3.1 Last Layer (Output)

The partial derivative of error with respect to the input is used to calculate the local gradient or error sensitivity at the output layer. In Eq. 3.8, L represents last layer, S represents weighted sum for specific neuron u, v, z at any layer. K, U, V, Z represents number of sample in the batch, $U = 1$, V is the number of output neurons, and $Z = 1$, respectively. U and Z are always 1 as it is 1D layer. t^k represents the target output at the final output layers for sample k.

$$\delta^L_{(u,v,z)} = \frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} \tag{3.8}$$

Using chain rule method, it become as:

$$\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} = \frac{\partial E}{\partial y^{L}_{(u,v,z)}} \ \frac{\partial y^{L}_{(u,v,z)}}{\partial S^{L,k}_{(u,v,z)}} \tag{3.9}$$

where

$$\frac{\partial y^{L,k}_{(u,v,z)}}{\partial S^{L,k}_{(u,v,z)}} = f_L^{'}(S^{L,k}_{(u,v,z)}) \tag{3.10}$$

MSE is the sum of the square difference between calculated output and target output and is given in Eq. 3.7. By putting Eq. 3.7 and 3.10 in Eq. 3.9, it will become as:

$$\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} = \frac{\partial (\frac{1}{K\times U\times V\times Z}\sum_{v=1}^{V}(y^{L,k}_{(u,v,z)} - t^{k}_{(u,v,z)})^2}{\partial y^{L,k}_{(u,v,z)}} \ f_L^{'}(S^{L,k}_{(u,v,z)}) \tag{3.11}$$

$$= \frac{2}{K\times U\times V\times Z}\left(y^{L,k}_{(u,v,z)} - t^{k}_{(u,v,z)}\right)\left(\frac{\partial y^{L,k}_{(u,v,z)}}{\partial y^{L,k}_{(u,v,z)}} - \frac{\partial t^{L,k}_{(u,v,z)}}{\partial y^{L,k}_{(u,v,z)}}\right) f_L^{'}\left(S^{L,k}_{(u,v,z)}\right) \tag{3.12}$$

As

$$e^k_{u,v,z} = \left(y^{L,k}_{(u,v,z)} - t^{k}_{(u,v,z)}\right) \tag{3.13}$$

and

$$\frac{\partial t^{k}_{(u,v,z)}}{\partial y^{L,k}_{(u,v,z)}} = 0 \tag{3.14}$$

Therefore,

$$\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} = \frac{2}{K \times U \times V \times Z}\, e^{k}_{(u,v,z)} \left(\frac{\partial y^{L,k}_{(u,v,z)}}{\partial y^{L,k}_{(u,v,z)}} - 0 \right) f'_{L} \left(S^{L,k}_{(u,v,z)} \right) \tag{3.15}$$

$$\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} = \frac{2}{K \times U \times V \times Z}\, e^{k}_{(u,v,z)} \ (1-0) \ f'_{L} \left(S^{L,k}_{(u,v,z)} \right) \tag{3.16}$$

$$\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}} = \frac{2}{K \times V}\, e^{k}_{(u,v,z)} \ f'_{L} \ (S^{L,k}_{(u,v,z)}) \tag{3.17}$$

As error sensitivity $\delta^{L,k}_{u,v,z}$ in Eq. 3.8 is calculated by $\frac{\partial E}{\partial S^{L,k}_{(u,v,z)}}$, therefore putting Eq. 3.17 in 3.8 and simplifying it will give:

$$\delta^{L,k}_{u,v,z} = e^{k}_{u,v,z} \ f'_{L} \ (S^{L,k}_{(u,v,z)}) \tag{3.18}$$

In order to normalize the term individually, we have to divide each by the sum of total number of neurons in all K samples, i.e. $K \times U \times V \times Z$ which can also be represented as $K \times N_L$ (to represent it the way it is represented in literature) for the outer layer or just $K \times V$ as the rest, i.e. U and Z are 1. It is given by Eq. 3.17. Also in our case, we did not normalized the output layer. Therefore, we can represent and calculate error sensitivity at final output layer in-terms of Eq. 3.18.

3.2.3.2 Full Connected Layer (L-1)

We have calculated the local gradient for the output neurons in previous section. Now to back-propagate them in fully connected 1D

layers, we do it in two steps as previously mentioned, i.e. calculating error sensitivities and then the error gradients. However, this is not simple as compare to output layer. As here, error is on output layer, and we have to transfer it through connections to each hidden neuron of 1D layer.

$$\delta_{u,v,z}^{l_n} = \frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} \tag{3.19}$$

Using Chain rule:

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{\partial E}{\partial y_{u,v,z}^{l_n,k}} \frac{\partial y_{u,v,z}^{l_n,k}}{\partial S_{u,v,z}^{l_n,k}} \tag{3.20}$$

Similar to previous layer, putting value of $\frac{\partial y_{u,v,z}^{l_n,k}}{\partial S_{u,v,z}^{l_n,k}}$ given in Eq. 3.10 and the error given by Eq. 3.7, we get:

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{\partial (\frac{1}{K \times V} \sum_{v=1}^{V} (y_{u,v,z}^{l_{n+1},k} - t_{(u,v,z)}^{k})^2}{\partial y_{u,v,z}^{l_n,k}} \quad f_{l_n}^{'}(S_{u,v,z}^{l_n,k}) \tag{3.21}$$

After simplifying the derivatives,

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V} \quad e_{u,v,z}^{k} \left(\frac{\partial y_{u,v,z}^{l_{n+1},k} - t_{(u,v,z)}^{k}}{\partial y_{u,v,z}^{l_n,k}} \right) f_{l_n,k}^{'} \left(S_{u,v,z}^{l_n,k} \right) \tag{3.22}$$

As, $l_{n+1} = L$ therefore we can not access error at final layer. We have to use chain rule once again.

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V} e_{u,v,z}^{k} \left(\frac{\partial e_{u,v,z}^{k}}{\partial y_{u,v,z}^{l_n,k}} \right) f_{l_n}^{'} \left(S_{u,v,z}^{l_n,k} \right) \tag{3.23}$$

Based on Eq. 3.13, it becomes:

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V} \; e_{u,v,z}^{k} \; f_{l_n}^{'} \left(S_{u,v,z}^{l_n} \right) \; \left(\frac{\partial e_{u,v,z}^{k}}{\partial S_{u,v,z}^{l_{n+1},k}} \frac{\partial S_{u,v,z}^{l_{n+1},k}}{\partial y_{u,v,z}^{l_n,k}} \right) \tag{3.24}$$

To simplify the above Eq. 3.24, as $l_{n+1} = L$ therefore, we can write L instead of l_{n+1} in the following equations. $\frac{\partial e^k_{u,v,z}}{\partial S^{l_{n+1},k}_{u,v,z}}$ can be further simplified as:

$$\begin{aligned}\frac{\partial e^k_{u,v,z}}{\partial S^{L,k}_{u,v,z}} &= \frac{\partial y^{L,k}_{u,v,z}}{\partial S^{L,k}_{u,v,z}} - \frac{\partial t^k_{(u,v,z)}}{\partial S^{L,k}_{u,v,z}} \\ &= f'_L\left(S^{L,k}_{u,v,z}\right)\end{aligned} \tag{3.25}$$

Similarly, $\frac{\partial S^{l_{n+1},k}_{u,v,z}}{\partial y^{l_n}_{u,v,z}}$ or $\frac{\partial S^{L,k}_{u,v,z}}{\partial y^{l_n,k}_{u,v,z}}$ can be further simplified as:

$$\frac{\partial S^{L,k}_{u,v,z}}{\partial y^{l_n,k}_{u,v,z}} = \frac{\partial\left(\sum_{v=1}^{V_L} w^L_{u,v,z}\; y^{l_n,k}_{u,v,z} + b^L_{u,v,z}\right)}{\partial y^{l_n,k}_{u,v,z}} = \sum_{v=1}^{V_L} w^L_{u,v,z} \tag{3.26}$$

or it can be written in three indices (however two of them are just 1 due to 1D vector) as:

$$\frac{\partial S^{L,k}_{u,v,z}}{\partial y^{l_n,k}_{u,v,z}} = \frac{\partial\left(\sum_{i=1}^{I} w^L_{i,v,z}\; y^{l_n,k}_{i,1,1} + b^L_{u,v,z}\right)}{\partial y^{l_n,k}_{i,1,1}} = \sum_{i=1}^{I} w^L_{i,v,z} \tag{3.27}$$

As $\frac{\partial t^L_{u,v,z}}{\partial S^{L,k}_{u,v,z}}$ is constant, so it's derivative is 0. Now we can put the resultant of Eq. 3.25 and 3.26 in Eq. 3.24, that gives:

$$\frac{\partial E}{\partial S^{l_n,k}_{u,v,z}} = e^k_{u,v,z}\; f'_{l_n}\left(S^{l_n,k}_{u,v,z}\right)\; \left(\; f'_L\left(S^{L,k}_{u,v,z}\right)\right)\; .\; \left(\sum_{i=1}^{I} w^L_{i,v,z}\;\right) \tag{3.28}$$

By re-arranging Eq. 3.28, we have found that $e^k_{u,v,z}\; f'_L\left(S^{L,k}_{u,v,z}\right)$ is given by Eq. 3.18 as error sensitivity at the upper layer (output) neuron. Further, as $\frac{\partial E}{\partial S^{l_n,k}_{u,v,z}}$ is represented by $\delta^{l_n}_{u,v,z}$ in Eq. 3.19, therefore, Eq. 3.28 becomes:

$$\delta^{l_n}_{u,v,z} = \;\; f'_{l_n}\;\left(S^{l_n,k}_{u,v,z}\right) \sum_{i=1}^{I} \delta^{l_{n+1}}_{i,v,z}\; w^{l_{n+1}}_{i,v,z} \tag{3.29}$$

Where l_{n+1} is the upper layer (output) and $'l'_n$ is the 1D fully connected layer. We have used i, j, m that are linked with current u, v, z.

As it is fully connected layer, therefore, we have only one summation in the equation to change the variable j (as other remains constant).

However to understand this easily, the best way is to consider these as simple 1D hidden layers of a multilayer perceptron as given below in Eq. 3.30.

$$\delta_n^{l_n,k} = \quad f'_{l_n} \ \left(S_n^{L,k}\right) \sum_{m=1}^{N_{l+1}} \delta_m^{l_{n+1},k} \ w_{m,n}^{l_{n+1}} \tag{3.30}$$

l_{n+1} is the upper layer (output) and $'l'_n$ is the 1D fully connected layer. Eq. 3.29 or 3.30 are the final equations to be used for calculating error sensitivity at fully connected layers. Now we have to calculate actual weight gradients to update our weights.

Weight Gradients at 1D or Fully connected layer:
The weight gradients of 1D fully connected layer is calculated by the product of local gradients calculated in Eq. 3.29 and their respective inputs that generated the output in the forward propagation. This is given in the following Eq. 3.31 (using traditional notations) and Eq. 3.32 (using 3 index notations).

$$\frac{\partial E}{\partial w_{m,n}^{l_n}} = \sum_{k=1}^{K} \sum_{m=1}^{M} \delta_n^{l_n,k} \ y_m^{l_{n-1},k} \tag{3.31}$$

In terms, of representing it in three variables, it becomes as given in Eq. 3.32. However, only one variable changes, the rest are 1 as it is 1D column vector.

$$\frac{\partial E}{\partial w_{u,v,z}^{l_n}} = \sum_{k=1}^{K} \sum_{j_{low}=1}^{j_{high}} \delta_{1,j,1}^{l_n,k} \ y_{u,v,z}^{l_{n-1},k} \tag{3.32}$$

Here $\delta_{1,j,1}^{l_n,k}$ represents the error sensitivity form the upper layer. Hence, it is a 1D vector therefore, i_{high} and m_{high} are equal to 1. It runs for only j_{high}, the number of neurons in the vector for the layers less than L, or the number of neurons in the output layer L, i.e. V. Whereas, K represents total number of sample frames in a batch.

This weight gradient is similar to calculating weight gradients of a fully connected layer in a multilayer perceptron.

Finally, Eq. 3.29 and 3.31 can be used for error sensitivity and error gradients at fully connected layers (between output and last pyramidal layer (L_P), i.e. $L_P < l_n < L$), respectively. Rather, even same equations are used for the L_P. However, the only difference is that in case of L_P after calculating error sensitivities and weight gradients, it is rearranged in 3D structure.

Bias gradient for 1D or Fully Connected Layer:
The biases are updated with the same error sensitivities. However, $\frac{\partial E}{\partial b^{l_n}_{i,j,d}}$ is calculated by

$$\frac{\partial E}{\partial b^{l_n}_{i,j,d}} = \sum_{k=1}^{K} \sum_{v_{low}=1}^{v_{high}} \delta^{l_n,k}_{u,v,z} \tag{3.33}$$

In 1D case, i and d are equal to 1, only j represents the number of output neurons for which their is one bias value. Therefore, the bias gradient is calculated by the summation of all the error sensitivities of that position in all the samples K.

3.2.3.3 3D Pyramidal Layer

Calculating error gradients at pyramidal layer is not similar to calculating gradients at *CNN*. The reason is the unique kernel for every neuron unlike *CNN* where same kernel is used for all the neurons. After calculating the error gradients at 1D layers, the error is back-propagated to update the weight parameters at 3D pyramidal layers. It starts in the same manner as for the other 1D layers. The difference arises after chain rule when the local error sensitivities of the higher layer are calculated.

$$\delta^{l_n,k}_{u,v,z} = \frac{\partial E}{\partial S^{l_n,k}_{u,v,z}} \tag{3.34}$$

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{\partial E}{\partial y_{u,v,z}^{l_n,k}} \frac{\partial y_{u,v,z}^{l_n,k}}{\partial S_{u,v,z}^{l_n,k}} \tag{3.35}$$

This becomes as:

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{\partial(\frac{1}{K \times V_L} \sum_{k=1}^{K_{l_n}} \sum_{v=1}^{V_{l_n}} (y_{u,v,z}^{l_n,k} - t_v^k)^2}{\partial y_{u,v,z}^{l_n,k}} f_{l_n}^{'}\left(S_{u,v,z}^{l_n,\ k}\right) \tag{3.36}$$

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V_L} e_v^k \left(\frac{\partial e_v^k}{\partial y_{u,v,z}^{l_n,k}}\right) f_{l_n}^{'}\left(S_{u,v,z}^{l_n,\ k}\right) \tag{3.37}$$

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V_L} e_v^k\ f_{l_n}^{'}\left(S_{u,v,z}^{l_n,\ k}\right) \left(\frac{\partial e_v^k}{\partial S_{u,v,z}^{l_{n+1},k}} \frac{\partial S_{u,v,z}^{l_{n+1},k}}{\partial y_{u,v,z}^{l_n,k}}\right) \tag{3.38}$$

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V_L} e_v^k\ f_{l_n}^{'}\left(S_{u,v,z}^{l_n,\ k}\right)$$

$$\left(\frac{\partial e_v^k}{\partial S_{u,v,z}^{l_{n+1},k}} \frac{\partial \left(\sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{d=d_{low}}^{d_{high}} w_{i,j,d}^{l_{n+1}} y_{i,j,m}^{l_n,k} + b_{i,j,m}^{l_{n+1},k}\right)}{\partial y_{u,v,z}^{l_n,k}}\right) \tag{3.39}$$

$$i_{low} = \left\lceil \frac{u - r_{l_{n+1}}}{g_{l_{n+1}}} \right\rceil + 1 \tag{3.40}$$

$$i_{high} = \left\lfloor \frac{u-1}{g_{l_{n+1}}} \right\rfloor + 1 \tag{3.41}$$

$$j_{low} = \left\lceil \frac{v - r_{l_{n+1}}}{g_{l_{n+1}}} \right\rceil + 1 \tag{3.42}$$

$$j_{high} = \left\lfloor \frac{v-1}{g_{l_{n+1}}} \right\rfloor + 1 \tag{3.43}$$

$\frac{2}{K \times N_L}$ is removed as it is used for normalization purpose, and we use other normalization techniques compare to this. Also due to mini batch approach, this is not used in our case. The value for lower and upper bounds, i.e. $i_{low}, i_{high}, j_{low}$, and j_{high} are calculated by Eq. 3.40, 3.41, 3.42, and 3.43, respectively. Whereas, 'm' represents the current map that is calculated by $d + z - 1$. Also, similar to the previous layer, using Eq. 3.27 or 3.26, gives us:

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V_L} e_v^k f_{l_n}^{'} \left(S_{u,v,z}^{l_n,\ k}\right) \left(f_{l_n}^{'} \ \left(S_{u,v,z}^{l_{n+1},k}\right) \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{d_{low}}^{d_{high}} w_{i,j,d}^{l_{n+1}} \right) \tag{3.44}$$

$$\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}} = \frac{2}{K \times V_L} f_{l_n}^{'} \left(S_{u,v,z}^{l_n,\ k}\right) e_v^k \left(f_{l_n}^{'} \ \left(S_{u,v,z}^{l_{n+1},k}\right) \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{d=d_{low}}^{d_{high}} w_{i,j,d}^{l_{n+1}} \right) \tag{3.45}$$

Hence, $\delta_{i,j,m}^{l_{n+1},k}$ is given by $e_v^k \ f_{l_n}^{'} \left(S_{u,v,z}^{l_{n+1},\ k}\right)$ and $\delta_{u,v,z}^{l_n,k}$ is given by $\frac{\partial E}{\partial S_{u,v,z}^{l_n,k}}$ in Eq. 3.34. Therefore,

$$\delta_{u,v,z}^{l_n} = f_{l_n}^{'} \ \left(S_{u,v,z}^{l_n,\ k}\right) \ . \sum_{d=1}^{D} \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{m=m_{low}}^{m_{high}} \delta_{i,j,m}^{l_{n+1},k} \ w_{i,j,d}^{l_{n+1}} \tag{3.46}$$

Where $u = 1, 2, 3 \ldots H_{l_n}$ and $v = 1, 2, 3 \ldots W_{l_n}$. In Eq. 3.46, i_{low}, i_{high}, j_{low} and j_{high} are calculated by Eq. 3.40, 3.41, 3.42, and 3.43, respectively. For each z, error sensitivity is calculated by respective maps m calculated in the range of $m_{low} = z$ and $m_{high} = z + D + 1$ from layer l_{n+1}. The temporal depth is $D = 3$. Eq. 3.46 is the final equation to calculate error sensitivities at 3D weighted sum/correlation layers.

Weight Gradients for 3D Pyramidal Layers:
The same procedure is followed as it is being done for 1D layer, i.e.

to calculate the weight gradients by taking the product of sum of local sensitivities at higher layer that are in contact with the current neuron at lower layer.

$$\frac{\partial E}{\partial\ w_{i,j,d}^{l_n}} = \sum_{k=1}^{K} \sum_{m=m_{low}}^{m_{high}} \left(y_{i,j,m}^{l_{n-1},k} \times \sum_{u=u_{low}}^{u_{high}} \sum_{v=v_{low}}^{v_{high}} \sum_{z=z_{low}}^{z_{high}} \delta_{u,v,z}^{l_n,k} \right) \quad (3.47)$$

here $i = 1, 2, 3, \ldots H_{l_n}$, $j = 1, 2, 3, \ldots W_{l_n}$, Whereas m is in the range given by m_{low} and m_{high} as shown below:
$m_{low} = d$
$m_{high} = M_{l_n} - (D - d)$

Where, 'D' represents temporal depth, i.e. 3, and d is equal to 1, 2 or 3. M is the number of total input frames/feature maps in that layer. However, z_{low} and z_{high} can be calculated by Eq. 3.52 and 3.53.

$$u_{low} = \left\lceil \frac{i - r_{l_n}}{g_{l_n}} \right\rceil + 1 \quad (3.48)$$

$$u_{high} = \left\lfloor \frac{i - 1}{g_{l_n}} \right\rfloor + 1 \quad (3.49)$$

$$v_{low} = \left\lceil \frac{j - r_{l_n}}{g_{l_n}} \right\rceil + 1 \quad (3.50)$$

$$v_{high} = \left\lfloor \frac{j - 1}{g_{l_n}} \right\rfloor + 1 \quad (3.51)$$

$$z_{low} = \left\lceil \frac{m - D}{G} \right\rceil + 1 \quad (3.52)$$

$$z_{high} = \left\lfloor \frac{m - 1}{G} \right\rfloor + 1 \quad (3.53)$$

'G' represents the number of frames that we leave after each 3D correlation. In our experiments it is kept as one, e.g. for first feature map, we take input maps 1,2, and 3, whereas for the second output map we consider 2,3, and 4. 'D' represents the depth as previously explained.

Bias gradient for 3D Pyramidal Layer:

The biases are also updated with the same error sensitivities. However, $\frac{\partial E}{\partial w_{i,j,m}^{l_n}}$ is calculated by

$$\frac{\partial E}{\partial w_{i,j,m}^{l_n}} = \sum_{k=1}^{K} \delta_{u,v,z}^{l_n,k} \tag{3.54}$$

The error gradient for a bias is the sum of all the error sensitivities of that position in all the maps from all the samples K. $b_{i,j,m}^{l_n}$ is the bias for neuron (i,j) in map m. As we have one bias for each output neuron, therefore $i = u$, $j = v$ and $m = z$.

3.2.3.4 3D Temporal Pooling Layer

The technique to calculate weight gradients is the same as 3D pyramidal Layer backpropagation. The only difference is that error is back-propagated through selected neurons only, e.g. if we use max pooling, than only the neuron having maximum value among the three receptive fields is updated.

$$\delta_{u,v,z}^{l_n,k} = \sum_{d=1}^{D} f_{l_n}^{'}\left(S_{u',v',z'}^{l_n,k}\right) \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \sum_{m=m_{low}}^{m_{high}} \delta_{i,j,m}^{l_{n+1},k} \; . \; w_{i,j,d}^{l_{n+1}} \tag{3.55}$$

Where the indices (u', v', z') represents the points when it attains the largest value in the receptive field $R_{u,v,z}^{l_n,d}$.

$$\underset{u',v',z'}{\operatorname{argmax}} S_{u',v',z'}^{l_n,k} := \{(u',v',z') | \forall (u,v,z) : S_{u,v,z}^{l_n,k} < S_{u',v',z'}^{l_n,k}\} \tag{3.56}$$

It represents the maximum of the maximum values among the three receptive fields calculated in the same manner as being done for

selecting the max value in the forward propagation by Eq. 3.2. $\left(S^{l_n,k}_{u',v',z'}\right)$ is the weighted sum value resulted from the weight parameter $(w_{u,v})$ and the max value.

The rest of ranges, i.e. $i_{low}, i_{high}, j_{low}$ and j_{high} are calculated by Eq. 3.40, 3.41, 3.42, and 3.43, respectively. For each z, error sensitivity is calculated by respective maps m calculated in the range of $m_{low} = z$ and $m_{high} = z + D + 1$ from layer l_{n+1}. The temporal depth is $D = 3$.

Weight Gradients for Pooling Layer: The weight gradients for 3D pooling layer are calculated by:

$$\frac{\partial E}{\partial w^{l_n}_{i,j}} = \sum_{k=1}^{K} \sum_{m=1}^{M_{l_n}} \left(y^{l_{n-1},k}_{i,j,m}\right) \cdot \sum_{u=u_{low}}^{u_{high}} \sum_{v=v_{low}}^{v_{high}} \sum_{z=z_{low}}^{z_{high}} \delta^{l_n,k}_{u,v,z} \tag{3.57}$$

where, $\frac{\partial E}{\partial w^{l_n}_{i,j}}$ are the weight gradients to be used in Eq. 3.6 to update the weight parameters for pooling layer. $\left(y^{l_{n-1},k}_{i,j,m}\right)$ is the max value calculated in Eq. 3.3. Range values ($v_{low}, v_{high}, u_{low}$, and u_{high}) are calculated by Eq. 3.48,3.49, 3.50 and 3.51, respectively. Eq. 3.52 and 3.53 are used for selecting corresponding maps, i.e. z_{low} and z_{high} containing error sensitivities. 'd' is always in the same range of $D = 3$.

Bias gradient for 3D pooling layer:

The biases are also updated with the same error sensitivities. However, $\frac{\partial E}{\partial b^{l_n}_{i,j}}$ is calculated by Eq. 3.58.

$$\frac{\partial E}{\partial b^{l_n}_{i,j,m}} = \sum_{k=1}^{K} \sum_{u=1}^{u_{high}} \sum_{v=1}^{v_{high}} \delta^{l_n,k}_{u,v,z} \tag{3.58}$$

The bias gradient calculation for pooling layer is almost same as 3D weighted sum/correlation layers. However, the difference is due to the reason that *3DPyraNet* have only one bias for each output map in pooling layer. Due to which $i = 1$ and $j = 1$ in Eq. 3.58. Therefore, the error gradient in case of biases is the sum of all the

error sensitivities of those maps for all the samples K. $b^{l_n}_{i,j,m}$ is the bias for all the neurons in that map m. u_{high} and v_{high} represents the total rows and columns in the feature map. Also, as *3DPyraNet* have only one bias for each map, therefore, in this case m is same as z.

In the next section, we discussed our results achieved after experimenting on two benchmark action recognition datasets.

3.3 Results & Discussion

To evaluate the performance of our *3DPyraNet*, rather than considering large datasets, we have decided to examine it with small but difficult dataset. The reason behind using these small scale datasets is two fold: first they are fast compare to very large datasets like UCF101 or HMDB1, second they are not favorable to most deep learning approaches due to lack of training data. Therefore, we evaluated *3DPyraNet* on *Weizmann* and *KTH* datasets [105, 16].

3.3.1 AR Evaluation Datasets

These datasets are small in sequences but each video consists quite similar and complicated action scenarios. We implemented our model in Matlab 2014b. We used their computer vision and machine learning tool boxes in some cases. The rest we coded our own implementation.

Weizmann:
It is a good starting dataset for evaluating performance of a network. It is smaller compare to others in terms of action sequences. However, it provides ten types of quite similar human actions i.e. Walking, Running, Jumping, Galloping sideways, Bending, One-hand wave, Two-hands wave, Jump in place, Jumping Jack, and Skip. Each action is done by nine actors in different scenarios that make it a complex task for recognition due to the reason that it have only few

short videos for each category, which is not a good scenario rather challenging for *DL* models.

KTH:

It is another popular big and complex dataset that contains six actions done by 25 actors. It provides 2391 sequences in four different scenarios along with camera movement that results in different resolutions. It is big compare to Weizmann dataset but have fewer action classes. However, one of the action class is very hard to be tracked by a tracker. As we need 13 consecutive frame for a sequence. Therefore, in some tests it is neglected for not providing or having consecutive enough proper segmented sequences. Therefore, in this case similar to many work being done in literature, the results are shown with 5 classes. We used a sequence of 13 consecutive frames of size $64x48$ to represent an action for both datasets.

3.3.2 Discussion

We carried out two type of experiments, i.e. 1) to check efficiency of proposed WS layers with simple activation functions and 2) combining it with pooling and using advance rectification functions. Therefore, for the first case we used a network with two WS layers and a fully connected layer to classify among ten classes. The output of each WS layer is passed through an activation function i.e. sigmoid or tangent and than normalized throughout the network learning. Initial learning is not smooth and it takes around 450 epochs to converge. This provide accuracy of 80% on training set and 70% on testing set.

As in most deep models, pooling plays an important role by providing translation invariance as well as reducing the dimensions. In addition, for faster convergence, avoiding local minima, and improvement in performance; an extension of rectified linear unit known as leaky rectified linear units ($LReLu$) [106] is used. This $LReLu$ in contrast to $ReLu$ shown in Eq. 3.59 and 2.18 respectively, allow a small non-zero gradient when the neuron is less than or equal to zero.

This property overcomes the limitation of $ReLU$ i.e. not updating weights if it stuck with zeros.

$$f(x) = \begin{cases} x & if x > 0 \\ 0.01x & otherwise \end{cases} \tag{3.59}$$

Therefore, in *3DPyraNet* model, we adopted pooling rather temporal 3D pooling represented by ($L2P$), and $LReLu$ in combination to WS layers. This results in high accuracy i.e. 87% and 80.5% respectively for training and testing along with faster convergence, i.e. within 200 epochs. In addition, learning behavior during training is quite smooth compare to the previous model. Further, when we use voting scheme for classification of videos based on classified action sequences, result increased by two to three percent.

3DPyraNet is being compared with deeper models that have five to eight hidden layers. To better evaluate our model, we reported the mean accuracy on five splits of training and testing datasets selected from same Weizmann database as done for evaluation of several other models.

We followed same recommended strategy for retaining train and test dataset. However, to cross validate the results, we randomized the data in same proportion keeping in mind that equal number of sequences should exist for the small number of sequences, e.g. 'skip' or 'running'. We achieved 90.9% accuracy for considering all ten classes in the dataset as shown in Tab. 3.3 (a). However, videos containing action 'skip' are very short. Many authors in literature didn't use this category in their experiments, e.g. [107]. Our model is unable to learn and classify skip category in proper manner due to lack of training data. In case of 'pjump', it faces the same problem of having less training sequences that results in poor performance. Therefore, if we neglect the skip category, accuracy increases to 92.46% as shown in Tab. 3.3 (b). However, for the rest of categories, *3DPyraNet* shows optimal results in both sequence and video classification shown on left of Fig. 3.6.

Results on Weizmann is comparable with the state-of-the-art model *3DCNN* reported in [60], which is impressive considering fewer number of hidden layers and having no sophisticated pre-processing for extracting hard coded features.

Table 3.3: (a) Mean accuracy of five random data setups, (b) Proposed Vs. Others for Weizmann and KTH datasets

(a)

SetUp	Accuracy(%)
1	90.5
2	92
3	90
4	90.5
5	91.3
Mean	90.9

(b)

Model(classes)	Weizmann (%)	KTH(%)	Layers
3DCNN	88.26[60]	90.2 [13]	6
3DPyraNet (all)	90.9	72	4
3DPyraNet (all-1)	92.46	74.23	4
ST-DBN	-	85.2	4
GRBM	-	90.0	-
Schuldt [16]	-	71.7	-
Dollar [108]	-	81.2	-
Alexandros [107](all)	90.32	-	-
Alexandros [107](all-1)	92.77	-	-

The second database used in our experiments is *KTH*. We used same criteria that took 9 out of 25 persons videos for testing as stated in literature. We randomly selected a total of 200 sequences from them having size of $13 \times 64 \times 48$. It should be noted that in our initial experiments, we faced the same problem for running videos, i.e. having less frames than minimum requirement of 13 due to fast movement of the person or camera zooming scenarios. We achieved 72% over six classes. If we remove running class due to less number of training data, our sequence accuracy becomes 74.23% (see confusion matrix on right of Fig. 3.6).

Tab. 3.3 (b) shows comparison of proposed model with state-of-the-art models reported in literature for Weizmann and KTH datasets. In case of Weizmann, we overcome reported best result of 88.26% with an average of 91.07% from ten tests in [60, 109]. We used same dataset and same number of consecutive input frames as used in [60, 109]. On the other hand for KTH dataset, *3DPyraNet* didn't show better result as provided by *3DCNN* [13], but still it shows comparable results to some of complex models. One of most plausible reason is that deep models need more data to have better understanding of their respective problem. Secondly, unlike *3DCNN* [13], where they use ROI sequences extracted and classified by another *CNN* based methodology. We use only background subtraction for extracting the human as ROI. This ROI may contain half, not in centered, or un-aligned person as input. This can greatly affect the learning process and may have high impact in reducing the classification rate compare to *3DCNN*.

Tab. 3.4 also shows result for *DSR* datasets, i.e. YUPENN and MaryLand-in-the-Wild. In addition it also shows number of parameters. It shows that *3DPyraNet* outperforms even in case of dynamic scene understanding/dynamic scene recognition (*DSU/DSR*) problem as well as in reducing number of parameters. However, these are discussed in detail in the coming chapter.

3.3.2.1 Computation Time

We are using Matlab version 2014b on an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz system. Our model with input batch size of $80 \times 100 \times 13 \times 100$ takes approximately 1.757034 minutes. Based on that, a clip of size $80 \times 100 \times 13$ takes approximately 1.05 seconds for complete forward and backward computation. The majority of time is taken by back-propagation in the pooling layer.

Table 3.4: Accuracies for Action (Weizmann and KTH) and Scene (YUPENN and MaryLand) datasets, Layers represents main layers, Parameters are in million, and size is in MB

Model(classifier)	Weizmann	KTH	YUPENN	MaryLand	Layers	Parameters in Millions (Size in MB)
3D-ConvNet [60]	88.26	89.40	-	-	7	0.01717 (0.31)
3DCNN [13]	-	90.2	-	-	6	0.00511 (0.09)
3DHOG [110]	84.3	91.4	-	-	-	-
Cuboids [111]	-	90	-	-	-	-
Gabor3D+HOG3D (SVM) [112]	-	93.5	-	-	-	-
3DSIFT (SVM) [113]	82.6	-	-	-	-	-
HOG+HOF+MBH +Trajectories (SVM) [114]	-	94.2	-	-	-	-
C3D (SVM) [23]	-	-	98.1	87.7	15	17.5 (305.14)
ImageNet [23]	-	-	96.7	87.7	8	17.5 (305.14)
ST-DBN [57]	-	85.2	-	-	4	-
Schuldt (SVM) [16]	-	71.7	-	-	-	-
Dollar (SVM) [108]	-	81.2	-	-	-	-
3DHOG+Local weighted SVM [115]	100	92.4	-	-	-	-
3DPyraNet	90.9	72	45	67	4	0.83 (14.58)
3DPyraNet (all-1)	92.46	74.23	-	-	4	0.83 (14.58)

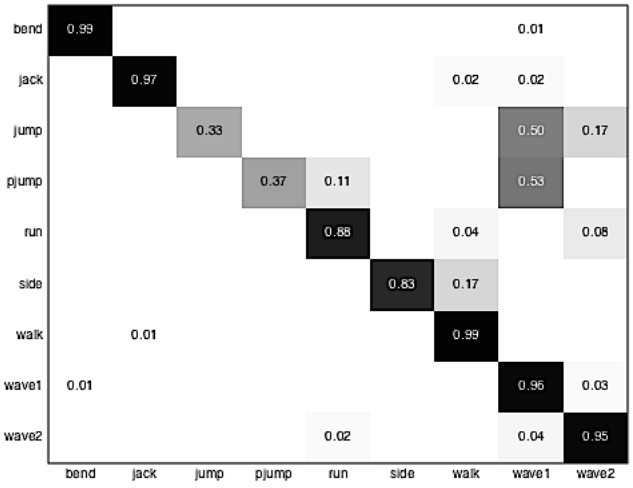
3.4 Chapter Summary

To conclude this chapter, we have proposed a generalized 3D pyramidal neural network architecture for recognition in videos. It gets raw input frames from videos as input. The approach is able to learn features in fewer layers due to pyramid structure. It provided better results in case of Weizmann and comparable results with KTH datasets. We are verifying the generality of our model by testing

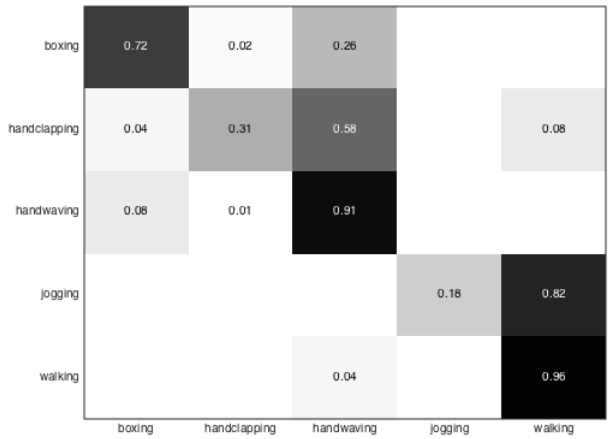
it on recent larger and challenging datasets like UCF sports, YU-PENN, MaryLand and UT-Interaction datasets. This will help in presenting benefits of using strictly pyramidal structure instead of non-pyramidal structure for learning a powerful model. Since, the model is aimed to obtain good performance despite the complexity and diversity of these datasets.

3.5 Related Publications

- Ihsan Ullah and Alfredo Petrosino. *A Strict Pyramidal Deep Neural Network for Action Recognition*, in Proceedings of International Conference on Image Analysis and Processing (ICIAP-15), pages 236-245, 07-11 Sep, 2015.
- Ihsan Ullah and Alfredo Petrosino. *A Deep Pyramidal Neural Network for Spatiotemporal Features Learning*, To be Submitted in IEEE Transaction on Neural Network and Learning System, 2016.



(a)



(b)

Figure 3.6: (a) Confusion Matrix for best case Weizmann without Skip (b) Current Best KTH without Running

Chapter 4

Spatiotemporal Feature Learning with *3DPyraNet*

Our Contribution:

- Presenting an effective approach for spatiotemporal feature learning using deep *3DPyraNet*.
- It analyzes spatial and temporal properties of a naturally occurring dynamic scene in a short video clip. The reciprocality of these properties is refined and preserved through all layers.
- Findings are twofold: 1) The proposed 3D weighting scheme is more suitable for feature learning in dynamic scenes captured by moving cameras compared to other hand crafted and feature learning approaches; 2) Our model shows improvement in both classification and containing a smaller number of parameters.
- In comparison to other recent techniques, our globally learned features with a linear classifier, result in competitive accuracy on first dataset and outperform on the second benchmark dataset.

4.1 Motivation

DSR is a key area of interest for automated video understanding. The capability to classify dynamic scene plays vital role in video surveillance, robot-navigation, video segmentation, etc. due to the reason that it provides key features about the presence or absence of an action, surface, or objects. As an example, a dynamic forest scene most probably comprise of static/non-static trees, animals or birds, whereas, a dynamic scene from a street mostly contains cars or pedestrians etc. moving in some specific direction [2]. Similarly, the presence of any movement or relation among the objects or actions may help to distinguish a specific scene. A set of dynamic patterns and their spatial layout with the passage of time describe a dynamic scene in a short recorded clip. For example, a combination of slowly moving clouds at the top from one side to the other, mid-scene water waves moving in forward and then backwash, and a foreground of static sandy texture describe a beach scene [2, 3, 4].

In the last decade, significant research has been done for scene recognition in still images. Several large scale datasets have been collected for better understanding of dynamic natural scenes. The SUN dataset contains 899 categories including 130,519 images [69]. Whereas, the Places dataset have 7,076,580 images from 476 scene categories and is the largest at the time of this writing [70]. Scene recognition from images involve classifying an image into one of the several given class categories. Traditional *CV* feature extraction such as *LBP, HOG, GIST*, and variations of *SIFT* have demonstrated good performance over previously mentioned datasets [69, 36, 116, 117]. (*DL*) approaches such as *CNN* have dominated image scene classification by obtaining high accuracy as compared to other state-of-the-art traditional approaches. Traditional *CV* and *DL* models have taken into consideration only the spatial description of the scene present in an image. Whereas, a dynamic scene classification in a video categorizes a specific scene according to the semantic labels

[117] derived from the scene. For example, a generic dynamic scene is classified 'whirlpool' if there occurs: a circular movement of water in a river or sea, causing a hole in the center towards the bottom where objects may be drawn, and not just based on the spatial attributes of the scene.

The recording of scene can be done with either stationary or moving cameras; thus, while scene motion is a feature, it is not exclusive of camera generated motion. In contrast, dynamic textures [118, 119] face problem due to complicated dynamic patterns, even in simple setup such as with a static camera and the peripheral field being totally occupied by the precise compound dynamic pattern [67]. Currently, there is an intense interest in spatiotemporal analysis at various levels of complexity, ranging from optical flow and dynamic texture analysis to high-level analysis in terms of scenes of particular events in a video. However, motion sometime establish relation with effects that can be investigated as problems or artifacts such as variations in light, specular effects etc. To deal with these problems, mainly two type of techniques are adopted, i.e. handcrafted descriptors or learned features in combination with a specific classifier.

Handcrafted descriptors used in the *CV* community such as *SOE*, *MSOE*, *HOF+GIST* or *Chaos+GIST* [3, 2, 67] or many others show good result for *DSR*. Despite advances in image recognition, classification of scene from one frame poses a unique challenge as it contains insufficient information for a scene recognition. The best way to enhance the performance is to incorporate temporal information in a framework. Some of the well-known handcrafted temporal models are reported in [67, 13, 19, 18].

In [120], proposed *SWLD* descriptor used simple classifier to achieve better performance than techniques that use sophisticated classifiers, e.g. *SVM*. Feature descriptor/extraction technique has more importance for achieving good results as compared to a sophisticated classifier in a recognition system. This importance is due to the discriminative power and less number of features extracted by the feature

descriptor that can be classified by any linear classifier. However, descriptors are limited with certain scenarios, e.g. orientation, texture, flow etc. Therefore, it is more beneficial to automatically learn all type of discriminative features from the input data that can be classified by any simple linear classifier such as *SVM, KNN* etc.

In this regards, recent *NN* based methods have been proposed by going deeper for learning more discriminative and varied features for scene recognition [23, 12, 24]. These deep models received great attention due to their huge success on large scale datasets [12] or based on the idea that they perform well for scene recognition from videos [23, 24]. D. Tran et al. in [23] also emphasized on the importance of learned features compared to a classifier. They showed that even with a simple linear classifier their *C3D* features report good results on different dynamic scene and action recognition datasets.

As previously discussed in Chapter. 2 and 3, an important aspect of convolutional *DL* models is their weight sharing concept. This scheme reduces the number of parameters compared to other conventional fully connected *NN* models, but increases the chance to reduce the discriminative power of the parameters, considering the huge amount of data from videos. For this reason, recent *DL* models increases the number of maps at each higher layer, resulting in large number of trainable parameters. In addition, it avoids faster convergence and result in over training. In its response, certain techniques are proposed such as dropout [12], or a sophisticated technique that learn connections rather than parameters in the fully connected layers [30], or re-parametarizing the matrix-vector multiplication in the fully connected layer while training the model [121]. Further, the increase in maps as the network goes deeper, does not maintain the strict/pure biological plausible pyramidal structure. 'Strict' mean reducing/refining feature maps / information and their size as the network goes deeper.

On the contrary, in the past, most models were all pyramidal and were following the biological plausible structure [11, 14, 33]. In our

previous work, we proposed *3DPyraNet* model that follows the biological concept of *NN* structure by doing refinement from layer one to the final layer. *3DPyraNet* model uses a new weighting scheme that extracts discriminative spatial and temporal information from the video. Therefore, we extend the model in two ways: 1) proposing an extension of the (*3DPyraNet*) model by enlarging the architecture to learn features from big raw input data; 2) in order to observe discriminative power of learned features, a separate classification layer is introduced to classify the learned features. Our model can be applied in a wide spectrum of application scenarios (with slight tuning) due to absence of requirements of the handcrafted features. In the next section, the extended model is explained. However, the *3DPyraNet* model is already explained in Chapter. 3, therefore, it is not explained in depth.

4.2 Proposed 3DPyraNet-F

Selecting an optimal architecture is a challenging problem since it depends on the specific application. A generalized model is shown in Fig. 4.1. It consists of two *3DCorr* layers, a *3DPOOL*, a *FC* layer, and a linear-*SVM* classifier layer. In coming subsections, the main layers are being discussed. Whereas, once convergence is achieved, features from the last *NORM6* layer are extracted and fused in a single column feature vector. This global/early fusion model is a balanced mix between the spatial and temporal information. These are incorporated in such a way that global information in both spatial and temporal dimensions are progressively accessed by the *SVM* layer. Finally, *SVM* layer trained model is used in testing phase to classify the feature vectors extracted using *3DPyraNet*. One-vs-All criteria is used for the classification of video clips.

4.2.1 Architecture

We call our model *3DPyraNet-F* because we fuse all the highest layer learned features provided by *3DPyraNet* into a single feature vector. The model is almost similar to *3DPyraNet* but the difference arises in three points:

- At an architecture level due to different input size.
- 'RF', and 'O' that result in a new big network architecture as well as due to the inclusion of the '*SVM*' classifier layer.
- Changing the learning parameter update rules for achieving faster convergence and better performance.

Each layer of the network is shortly explained here.

3D Correlation Layer:

A dynamic scene can be recognized in a frame by similar structure or objects, e.g. a beach scene might be recognized by combination of moving clouds, waves, static sandy texture, and may have some human beings playing or lying on the sand. Therefore, a correlation layer at the beginning is most appropriate for learning the similarity present in small receptive fields of consecutive frames or clips of a scene in a video. Note that in this model we changed the notations for each layer, e.g. weighted sum layer (*L1WS*) is represented by '*3DCorr1*'. However, the mechanism is the same.

The first phase of *3DPyraNet-F* uses its 3D structure to incorporate the spatial as well as temporal information from the given input frames. The 3D weight matrix is of equal size to the input image/feature map where for each output neuron we have a fixed small unique 3D kernel (as discussed in previous Sec. 3.2.1). It performs *Corr* operation among the given neurons and their respective weights in a receptive field of a frame and corresponding weight matrix as shown in Eq. 4.1. This weight matrix generates sparse features as compared to a convolutional kernel. It is trained using

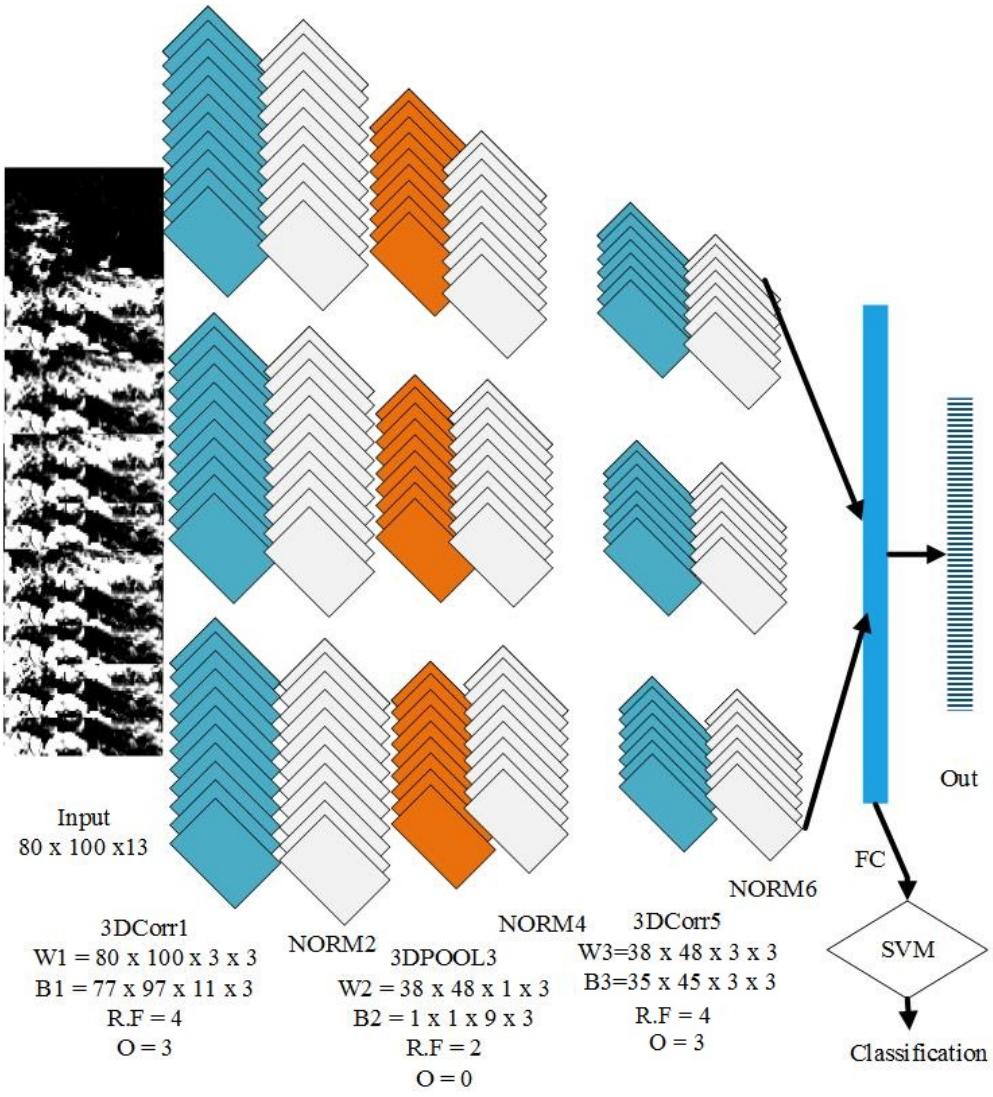


Figure 4.1: Proposed model 3DPyraNet-F (It becomes simple *3DPyraNet* if we remove *SVM*). Blue represents Correlation layers (weighted sum), gray represents normalization, brown represents pooling, and bright blue represents fully connected layer

updated *BP* (explained and derived in Sec. 3.2.3) with mini-batch *SGD* approach.

Several sets of these 3D matrices are used to extract varied features. We randomly initialize these weight parameters on each layer. However, we take care of recommended initialization techniques stated in literature [122, 123]. The activated resulting map is normalized before passing it as input to the next layer. The normalization not only helps in enhancing accuracy but also helps in faster convergence of the network.

$$y^{l_n}_{u,v,z} = f_{l_n}\left(\sum_{d=1}^{D}\sum_{(i,j,m)\in R^{l_n,d}_{(u,v,z)}}\left(\left(w^{l_n}_{(i,j,d)} \cdot y^{l_{n-1}}_{(i,j,m)}\right) + b^{l_n}_{(u,v,z)}\right)\right) \quad (4.1)$$

here f_{l_n} represents activation function used at current layer l_n. $(y^{l_n}_{u,v,z})$ directs us to output neuron location at $'z'$, where (u, v, z) represents the current neuron (u, v) at output map that is generated by set of input maps (represented by $'m'$) in temporal direction. $'m'$ is calculated by $(d + z - 1)$ from layer $'l'_{n-1}$ as shown in Eq. 4.2 third row. $W^{l_n}_{(i,j,d)}$ represents current weight parameter at position (i, j) of matrix $'d'$. In our experiments, at each layer we have $d_{low} = 1$ and $d_{high} = D$ where 'D' is the kernel temporal depth, i.e. $D = 3$. The set of neurons of a receptive field, i.e. (i, j) in the current map $'m'$ at lower layer is calculated by Eq. 4.2. Where, $R^{l_n,m}_{(u,v,z)}$ represents receptive field for each neuron (u, v) in $'z'$ output map. Here, r_{l_n} in the Eq. represents the size of receptive field, i.e. $'RF'$ at that layer l_n. $b^{l_n}_{(u,v,z)}$ represents the bias for the current layer.

$$R^{l_n,d}_{(u,v,z)} = \left\{\begin{array}{c}(i,j,m) \mid (u-1)+1 \le i \le (u-1)+r_{l_n}; \\ (v-1)+1 \le j \le (v-1)+r_{l_n}; \\ (d_{low}+z-1) \le m \le (d_{high}+z-1)\end{array}\right\} \quad (4.2)$$

Unlike *CNNs*, we does not use one bias for each output feature map, rather we use one bias for each neuron in an output feature map.

Further, similar to *3DCNN*, our model also extracts more type of features from the same spatial position of the input image/map by using multiple 3D weight sets. However, a *3DCNN* increase their set of kernels at each higher layer, whereas, a *3DPyraNet-F* keeps it fixed after starting layer and at each upper layer, the maps are decreased by two maps. Initialization with multiple kernels help our model not only in increasing different types of feature maps, but also, as a consequence of continuous reduction of maps, maintain a pyramidal structure of our model. Consequently, the model have less number of refined feature maps and parameters at the higher layer that help in faster convergence and avoiding over training.

(RF) and (O) are tuned for handling the performance. In this model, we use size of *(RF,O)* as (4,3) and (4,3) in *3DCorr1* and *3DCorr5*, respectively. After passing the feature maps through an activation function, same regularization approach of saturation with simple zero mean and unit variance is applied on the feature maps. This approach of normalization helps in faster convergence and increase in performance.

3D Temporal Pooling Layer The weight matrix approach has a slight deficiency of not learning translation and scale invariant features. Therefore, *3DPyraNet* introduced a temporal pooling layer to overcome these limitations. *3DPyraNet* performs $3D$ temporal pooling that is represented by *3DPOOL*. *3DPOOL* helps in learning translation and scale invariant features. It also helps in reducing the dimensionality not only in spatial domain but also in the temporal domain. This spatio-temporal reduction help *3DPyraNet-F* in maintaining its pyramidal structure. Max-pooling is used as it returns the maximum of the maximum values of the three receptive fields. It is helpful in removing non-maximum values that reduce computation on higher layers as well as provide translation invariance and robustness.

Traditional pooling layers in a deep *CNN* does not contain weight and bias parameters. However, our model consists of a weight pa-

rameter for each output maximum value among the three referenced fields. Each maximum value is multiplied with a weight parameter and then added with a bias. In this layer, we have only one bias for each output feature map. In the end, the resultant value is passed through an activation function as shown in Eq. 4.3.

$$y_{u,v,z}^{l_n}=f_{l_n}\left(\left(w_{u,v}^{l_n}\ .\ \max_{1\leq d\leq D}\left(\max_{(i,j,m)\in\ R_{u,v,z}^{l_{n-1},d}}\left(y_{i,j,m}^{l_{n-1}}\right)\right)\right)+\ b_{u,v,z}^{l_n}\right) \tag{4.3}$$

In Eq. 4.3, $R_{u,v,z}^{l_n,m}$ calculates the range for (i,j,m) indices, i.e. i_{low}, j_{low}, i_{high}, j_{high}, m_{low} and m_{high} as being calculated by Eq. 4.2 in previous layer. The size of the receptive field (RF also represented as 'r_{l_n}' in Eq. 4.2) is different than the receptive field of *3DCorr* layer. *(RF,O)* is taken as (2,0) in layer *3DPOOL3*.

Fully Connected Layer Output of correlation *(3DCorr5)* layer is normalized at *(NORM6)* layer. The resultant normalized discriminative feature maps are converted into a 1D column vector that consist of motion information encoded in multiple adjacent frame. It is used as a fully connected layer for classification. It can be extended to multiple 1D FC layers, depending on the complexity of the target application area. In addition, the size of this vector depends on the input size, total number of layers (L), RF, O, and D.

$$y_{u,v,z}^{l_n}=f_{l_n}\left(\sum_{j=j_{low}}^{j_{high}}\left(\left(W_{(i,j,m)}^{l_n}\ .\ y_{(i,j,m)}^{l_{n-1}}\right)+b_{(u,v,z)}^{l_n}\right)\right) \tag{4.4}$$

Eq. 4.4 is similar to Eq. 4.1. However in 1D case the u and z are 1. Only v is more than one. Therefore, $((i,d)_{low},(i,d)_{high})$ are 1 due to u and z being 1. If $l_n = L$ than it means that it is the final output layer otherwise, it is a 1D fully connected layer that is given as input to calculate output. Weight update is done based on the equations derived in previous chapter Sec. 3.2.3.

Linear SVM Layer:
3DPyraNet-F works mainly in two stages. First stage is to train it similar to *3DPyraNet*. Once its convergence is achieved, in the second stage, features from the last *NORM6* layer are extracted and fused in a single column feature vector. These are incorporated in such a way that global information in both spatial and temporal dimensions are progressively accessed by the *SVM*. Finally, *SVM* layer trained model is used in testing phase to classify the feature vectors extracted from testing set. One-vs-All criteria is used for the classification of video clips.

4.2.2 3DPyraNet-F Training

A fast training algorithm must be devised to learn recognition task efficiently. The objective of *3DPyraNet-F* training is to gradually minimize an error function that is defined in terms of our network outputs and their respective desired outputs. It is similar to *3DPyraNet*. It uses the same *CE* error function. Therefore, we adopt *CE* that calculates the posterior probabilities membership for each scene category. Delta rule given in Eq. 4.5 is used to update the weight parameters.

$$w_{u,v,d}^{l_n,new} = w_{u,v,d}^{l_n,old} - \varepsilon \frac{\partial E}{\partial w_{u,v,d}^{l_n}} \tag{4.5}$$

Where ε is the learning rate that controls the oscillation during training. We start with $\varepsilon = 0.000015$ and reduce it by multiplying it with 0.9 after each 4 epochs. Batch size of 100 is used in all experiments.

4.2.3 3DPyraNet-F_M

The difference between *3DPyraNet-F* and *3DPyraNet-F_M* is in the construction of feature vectors. After the network convergence, rather than just concatenating all the feature in one column vector, this model first converts feature maps of the same set in one single column vector. Then, the resultant feature vectors are summed together

Model	*3DCORR*	*3DPOOL*	*3DCORR*	*FC*	Output
3DPyraNet-F	$61 \times 45 \times 11 \times 3$	$30 \times 22 \times 9 \times 3$	$27 \times 19 \times 7 \times 3$	10773	6/10
3DPyraNet-F_M	$61 \times 45 \times 11 \times 3$	$30 \times 22 \times 9 \times 3$	$27 \times 19 \times 7 \times 3$	3591	6/10
3DPyraNet-F	$77 \times 97 \times 11 \times 3$	$38 \times 48 \times 9 \times 3$	$35 \times 45 \times 7 \times 3$	33075	13/14

Table 4.1: Network Structure used for Action (Weizmann(10) and KTH(6) shown in first two rows) and Scene (MaryLand (13) and YUPENN(14)) datasets. Feature map size at main Layers is shown for each model as well as the number of output classes

and divided by the number of weight sets to derive their mean vector. This results in a smaller feature vector compared to the previous model resulting in faster processing. These features have a local impact due to their addition with other features maps. The rest of the model and network architecture is similar to *3DPyraNet-F*.

Beside these two models, the third variation of *3DPyraNet-F* can be the size of the network. This difference exists due to different size input images of *AR* and *SR* datasets. Network structures for all models are given in Tab. 4.1.

4.3 Results & Discussion

We analyze the performance of *3DPyraNet-F* with both *AR* and *DSR* tasks. The networks and their sizes being used are shown in Tab. 4.1. We evaluate *AR* with same KTH and Weizmann datasets. Whereas, *DSR* with the help of *YUPENN* and *MaryLand-in-the-wild* dataset. These datasets are discussed in coming sub-section whereas datasets for *AR* are already being discussed in previous chapter Sec. 3.3.1. Training is done in the same way as done for simple *3DPyraNet* (already discussed in previous section). However, in this model, the learning rate is updated more frequently.

Each model is trained on its respective dataset. Tab. 4.1 shows a feature map size in the form of $w \times h \times m \times s$. Where 'w', 'h', 'm' and 's' represents width, height, number of maps, and weight sets, respectively. *KTH* and *Weizmann* have similar input size, i.e. $64 \times 48 \times 13$ with leave one frame out. Whereas, *YUPENN* and *MaryLand*

has $80 \times 100 \times 13$ with an overlap of 7 images for each clip. Training is done by SGD with mini batch size of 200 and 100 clips for AR and SR datasets, respectively. Further, an additional key advantage of the proposed model (i.e. it's fewer trainable parameters) is discussed in the end of this section.

4.3.1 DSR Evaluation Datasets

Dynamic scenes are categorized by a collection of dynamic patterns and their spatial layout, as recorded in small video clips. For example, a beach scene might be characterized by water waves and at the front a static sandy texture. Other examples include iceberg collapse, volcano eruption, whirlpool, boiling water and traffic scenes. These dynamic scenes may be taken with either static or moving cameras; thus, while scene motion is a characteristic, it is not separate from camera induced motion. Indeed, dynamic scene recognition from moving cameras has proven to be more challenging as compare to static cameras. *YUPENN* and *MaryLand* (also referred as 'MaryLand-in-the-wild') are two *DSR* benchmarks recorded with a static and a moving camera, respectively.

4.3.1.1 YUPENN

This dataset consists of short time videos having emphasizes on scene specific temporal information. It does not contain camera induced motion, rather the videos are stabilized. It consists of 14 scene categories. It consists of 420 videos having 30 videos of same size per class, i.e. beach, city street, elevator, forest fire, fountain, highway, lighting storm, ocean, railway, rushing river, sky-clouds, snowing, waterfall, and windmill farm. The number of frames and their size in a video is $250 \times 370(pixels) \times 145(frames)$. They are taken with a Canon HFS20 camcorder and online video repositories, such as YouTube, BBC Motion Gallery and Getty Images.

4.3.1.2 MaryLand-in-the-wild

Similarly, *MaryLand* consists of 130 videos of 13 scene categories collected from video hosting websites like 'Youtube'. The videos are recorded in totally uncontrolled environment, therefore the dataset has large variations in terms of illumination, rate, view, scale and camera dynamics. Each class contains 10 videos of varying size. These categories are Avalanche, Boiling water, Chaotic_Traffic, Forest_Fire, Fountain, Iceberg_collapse, Land_Slide, Smooth_Traffic, Tornado, Volcano_Eruption, Waterfall, Waves and Whirlpool. Fig. 4.2 shows sample images from *YUPENN* and *MaryLand* dataset. Both of these datasets are quite complex having similar appearance even for different classes as well as large difference even in the same categories.

One point to highlight is that there is large intra-class variation in these classes. Without temporal information, randomly chosen frames from these videos may cause confusion, e.g. 'chaotic_traffic' and 'smooth_traffic', 'avalanche' and 'iceberg', etc. Similarly, only using temporal information without use of spatial information may cause confusion between classes like, e.g. 'avalanche' and 'landslide', or 'tornado' and 'whirlpool', etc. This make it a difficult task for many algorithms to classify these videos in their respective categories.

4.3.2 AR Performance

KTH and *Weizmann* are easy for an in-depth study due to less data and more classes, however, challenging as well due to less data for training a deep model. Features form the *Norm6* layer of our trained model are extracted and fed to a linear-*SVM* classifier. It classifies each class similar to what is done in [124, 16, 108]. However, we perform two types of extraction, i.e. local and global fusion of features as in [24]. In first case, i.e. *3DPyraNet-F* feature vector becomes 10733. Whereas, in second case, i.e. *3DPyraNet-F_M*, the dimen-

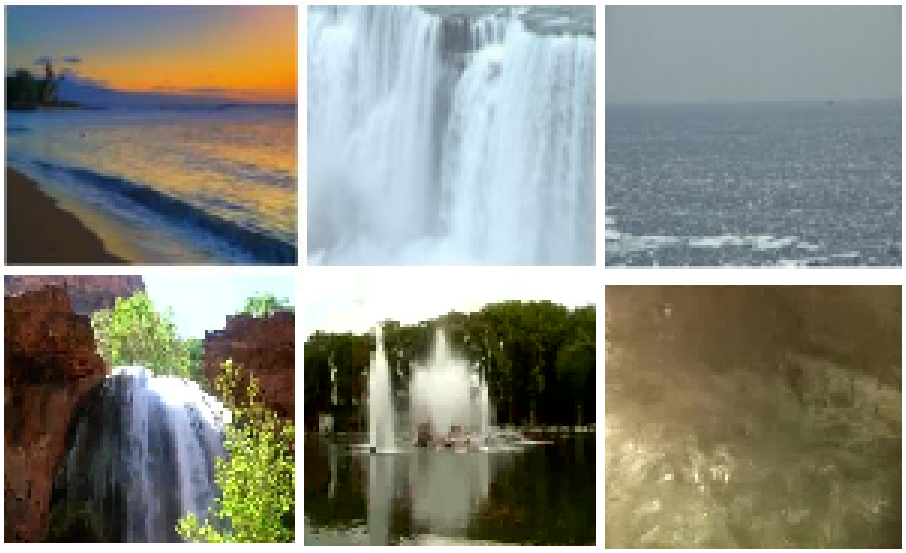


Figure 4.2: Samples images from YUPENN (1^{st} row from left to right (Beach, Waterfall, Ocean)) and MaryLand (2^{nd} row from left to right (Waterfall, Fountain, Boiling_Water)

sion of vector consist of 3591 features. *3DPyraNet-F* achieved mean accuracy of 93.42% from the binary classifications of one-vs-all scenario. Further, (*3DPyraNet-F_M*) enhances the overall performance by 0.67%. Similarly to [16, 13, 108], despite fewer training examples, global fusion (*3DPyraNet-F*) achieved optimal accuracy. In comparison to handcrafted features, our learned feature with *SVM* gets better results than *3DHOG*, *Cuboids*, and *Gabor3D+HOG3D*, whereas, almost equal performance is achieved when compared to the combination of *HOG*, *HOF*, *MBH*, and *Trajectories* descriptors [114], highlighting more discriminative power of our learned features.

We adopt the trained model on full *Weizmann* dataset and preprocess it as we did for *KTH*. The *3DPyraNet-F* and *3DPyraNet-F_M* models are applied. However, despite more classes, optimal results are achieved compare to state-of-the-art models as shown in Tab.4.2. *(SVM)* in Tab.4.2 shows that this model uses binary *SVM* classifier for one-vs-all classification. *3DPyraNet-F* enhances previous results of *3DPyraNet* for *Weizman* by 8.09% whereas, *3DPyraNet-F_M* en-

hances it further with an additional 0.14%. Only in the case of combination of (*HOG + HOF + MBH + Trajectories*), *3DPyraNet-F_M* have a lower accuracy of 0.87%. Similarly, *3DPyraNet-F* increases the accuracy for *KTH* dataset by 20% which is further increased (0.67%) by *3DPyraNet-F_M*.

4.3.3 DSR Performance

In [2], rather than using a filter over the whole image they use it over sub-regions to maintain and understand the spatial layout of the scene. Similarly, our model learns the whole mask rather than a specific portion of the image. As discussed in Sec. 3.2, *3DPyraNet* weight matrix is of equal size to input image/feature map. Therefore, it could be an ideal case for maintaining spatiotemporal layout for scene recognition in videos. It can also be used as a hint in other recognition tasks. The model for these datasets has a bigger input size compared to previous datasets of action recognition, i.e. $80 \times 100 \times 13$ resulting in feature vector of size 33075. We have taken an overlap of 7 frames, considering a small number of frames compared to previous models [23, 2, 19] - for instance, [23] uses $128 \times 171 \times 16$ frames in a clip from which $112 \times 112 \times 16$ random crops are extracted for data augmentation purpose.

In case of the *YUPENN* dataset, our model achieves a better accuracy of 96.2134% after 25 epochs. However, it achieves mean accuracy of 93.67% for a one-vs-all classification. Although, its accuracy is better at huge margin than the accuracy achieved by models in [2, 125, 18], still it does not report state-of-the-art performance by 5.33% less accuracy compare to model in [67] as shown in Tab. 4.2. The reason could be that the state-of-the-art model [67] combines several complex pre-processing and feature encoding techniques (PCA, LLC, GMM, IFV, static pooling and their proposed dynamic spacetime pyramid pooling in SPM) resulting in achieving higher accuracy than the one achieved by deep *C3D* model. Further, a worth mentioning fact is that the *C3D* [124] is pre-trained on Sports 1-

Million videos dataset [24], having high resolution and more frames in a clip, and using data augmentation (as previously mentioned). Whereas, we train our model on the same small dataset, having smaller image size and less number of frames in a clip, and without data augmentation in order to show that a good architecture can learn and give good performance, despite limited data. Although, our model is less competitive compare to *C3D* and Imagenet, still *3DPyraNet-F* achieves comparable results on a smaller dataset, i.e. 93.67%; a good starting point for future work to test *3DPyraNet-F* on very large scale dataset. Christoph et al. model [19] performance, as shown in Tab. 4.2 is better than ours by 1.5% but their results are based on majority voting for video classification. Unlike ours, where we classify each clip individually.

In order to further evaluate strength of *3DPyraNet-F*, unlike *YUPENN*, we test it with a more complex dataset, i.e. *MaryLand* that include camera motion. Despite camera motion, we achieve state-of-the-art accuracy of 94.83% as shown in Tab. 4.2, highlighting the discriminative power of the proposed model. *3DPyraNet-F* outperforms the state-of-the-art learned *C3D* features [124] by 7.13%. Whereas, the state-of-the-art model [67] by 14.83% (in-terms of *YUPENN* dataset). Classes such as boiling_water, fountain, iceberg_collapse, whirlpool show slight poor performance. One of the reason could be that all these categories contain some sort of similarity, i.e. water which make them ambiguous for correct classification. The performance of *3DPyraNet* (multiclass) in Tab. 4.2, i.e. 45% for *YUPENN* dataset shows that when the features are not learned properly, the classifier does not give good performance.

Tab. 4.3 shows a comparison per class between our model and the state-of-the-art feature learner and handcrafted model. Some of the models achieved 100% accuracy, but in other categories our model outperformed them. Similarly, Tab. 4.4 shows per class comparison of our model with the state-of-the-art on *YUPENN* dataset. One of the surprising factors that we analyze is that *3DPyraNet-F* shows

better performance when there is camera movement, which is quite unfamiliar. The reason for this behavior could be that our model weighting and the pyramidal scheme can learn motion from camera dynamics more appropriately. It is also considered that the current model architecture may not be fully tuned for *YUPENN* dataset. Therefore, it is expected that further tuning of the model may perform better for *YUPENN* dataset.

4.3.4 Parameters Reduction

After the strong success of ImageNet [12, 23], models became deeper that resulted in a large number of trainable parameters. A separate consideration should be made about the reduction of parameters. The number of parameters is unarguably a substantial issue in application space and the memory cost increases due to the large size of trained models on the disk [12, 23, 30, 96].

Network in Network (*NiN*) [96] highlighted the issue of reducing parameters, but they achieved it at a greater computation cost. As most of the parameters are in fully connected (*FC*) layers, C. Szegedy et al. [31] uses sparsity reduction complex methodologies for refining those trained models. S. Han et al. [30] tries to learn connections in each layer instead of weights and then the network is trained again to reduce the number of parameters. Whereas, Z. Yang et al. [121] in the deep fried network reparametarize their fully connected layer matrix-vector multiplication. This technique results in reducing memory and computational cost, i.e. from $O(nd)$ to $O(n)$ and $O(nlogd)$. *3DCNN* model have less parameters than ours, but based on the performance and lack of preprocssing give an edge for our model. Secondly, *C3D* model has about $17.5M$ parameters. Due to our proposed 3D weighting scheme, however, our model has less than a million parameters, i.e. 0.83M parameters for *YUPENN* and *MaryLand* datasets. Disk occupancy is almost negligible compared to the model trained by *C3D*; this is of great help in embedded systems and mobile devices where the memory usage is a problem.

4.4 Chapter Summary

An extension of 3D pyramidal neural network approach has been proposed that learns spatiotemporal features from raw gray level input frames of a given video clip. Due to the proposed weighting scheme, the number of parameters are far less in our model as compare to other deeper models. In addition, it is ideally suitable for learning scenes due to the equal size of an input image/frame and a weight matrix that preserves spatio-temporal layout. Further, we show that our fusion model is capable of learning powerful motion features by refining the sparse learned features, achieving competitive results with respect to current best methods on different video analysis benchmarks for scene recognition. We show that even with limited data, a good architecture can achieve competitive accuracies. Furthermore, a dynamic scene recognition system can provide relevant contextual information in many other applications of video analysis, e.g., scene context can improve human action recognition. In the future, we are verifying the generality of our model by going deeper and testing it on the same, as well as more recent larger and challenging datasets.

4.5 Related Publications

- Ihsan Ullah and Alfredo Petrosino. *Spatiotemporal Features Learning with 3DPyraNet*, Accepted in Advanced Concepts for Intelligent Vision Systems (ACIVS-16), pages 236-245, 24-27 Oct, 2016.
- Ihsan Ullah and Alfredo Petrosino. *A Deep Pyramidal Neural Network for Spatiotemporal Features Learning*, To be Submitted in IEEE Transaction on Neural Network and Learning System, 2016.

Table 4.2: Accuracies for Action (Weizmann and KTH) and Scene (YUPENN and MaryLand) datasets, Layers represents main layers, Parameters are in million, and size is in MB

Model(classifier)	Weizmann	KTH	YUPENN	MaryLand	Layers	Parameters in Millions (Size in MB)
3D-ConvNet [60]	88.26	89.40	-	-	7	0.01717 (0.31)
3DCNN [13]	-	90.2	-	-	6	0.00511 (0.09)
3DHOG [110]	84.3	91.4	-	-	-	-
Cuboids [111]	-	90	-	-	-	-
Gabor3D+HOG3D (SVM) [112]	-	93.5	-	-	-	-
3DSIFT (SVM) [113]	82.6	-	-	-	-	-
HOG+HOF+MBH +Trajectories (SVM) [114]	-	94.2	-	-	-	-
C3D (SVM) [23]	-	-	98.1	87.7	15	17.5 (305.14)
ImageNet [23]	-	-	96.7	87.7	8	17.5 (305.14)
ST-DBN [57]	-	85.2	-	-	4	-
Schuldt (SVM) [16]	-	71.7	-	-	-	-
Dollar (SVM) [108]	-	81.2	-	-	-	-
3DHOG+Local weighted SVM [115]	100	92.4	-	-	-	-
3DPyraNet	90.9	72	45	67	4	0.83 (14.58)
3DPyraNet (all-1)	92.46	74.23	-	-	4	0.83 (14.58)
3DPyraNet-F	98.99	93.42	93.67	**94.83**	4	0.83 (14.58)
3DPyraNet-F_M	**99.13**	**94.083**	-	-	4	0.83 (14.58)
Christoph (SVM) [67]	-	-	**99**	80	-	-
Christoph (SVM) [19]	-	-	96.2	77.7	-	-
Christoph (SVM) [18]	-	-	86.0	67.7	-	-
Theriault (SVM) [125]	-	-	85.0	74.6	-	-

Table 4.3: *MaryLand* dataset per class Accuracies vs state-of-the-art models

Class	C3D [124, 67]	DPCF [67]	*3DPyraNet-F*
Avalanche	90	90	93
Boiling Water	90	60	97
Chaotic Traffic	90	100	97
Forest Fire	80	90	90
Fountain	60	80	99
Iceberg Collapse	60	50	97
Landslide	70	80	96
Smooth Traffic	80	70	98
Tornado	80	80	91
Volcanic Eruption	90	90	94
Waterfall	40	70	98
Waves	100	100	97
Whirlpool	80	80	85
Overall	78	80	95

Table 4.4: *YUPENN* dataset per Class Accuracies vs state-of-the-art models

Class	C3D [124]	DPCF [67]	*3DPyraNet-F*
Beach	97	100	92
Elevator	100	100	94
Forest Fire	100	97	94
Fountain	83	93	93
Highway	97	100	93
Lightning Storm	93	100	99
Ocean	100	100	98
Railway	97	100	93
Rushing River	100	100	100
Sky-Clouds	97	100	86
Snowing	97	97	93
Street	100	100	93
Waterfall	97	97	93
Windmill Farm	100	100	90
Overall	97	99	94

Chapter 5

Adopting Strictly Pyramidal Architecture in Deep CNN

Our Contribution:

- Reporting a strict pyramidal convolutional neural network based on the concept of biological and image pyramid structure.
- We introduced a generalized statement over convolutional layers from input till fully connected layer that helps further in understanding deep architecture.
- It can be applied to all deep models if it provides pyramid structure in reverse that will result in reducing ambiguity, number of parameters, and their size on disk without degrading overall accuracy.
- In comparison to other recent techniques, our model result in competitive accuracy on five benchmark datasets.

5.1 Motivation

CNN based models play a significant role in raising deep learning society. However, all these models use the same concept of producing feature maps in convolutional layer followed by a pooling layer

to reduce the dimension of the map. Models such as *Alexnet* [12], *GoogleNet* [31], *VGG* [126] and many others [58, 127, 128, 129] [130] asserts that the deeper you go, the more the network performs well. In addition to going deeper, the models slightly changed the concept for avoiding vanishing of gradients by using class inference in consecutive convolution layers and max pooling layer, or using a layer wise softmax layer that re-boosts the vanishing gradients [31, 126, 62]. Others used new activation functions, weight updates regularization methods, class inferences, layer wise pre-training in supervised approaches which showed very promising results [127, 130].

Increasing number of layers means increasing number of parameters in a network. With the introduction of Network in Network (*NIN*) model [96], the issue of reducing parameters is further highlighted once again. However, it tends toward greater computation. As most of parameters exist in fully connected (*FC*) layers, therefore, C. Szegedy et al. [31] use sparsity reduction complex methodologies for refining the trained models. S. Han et al. [30] tried to learn connections in each layer instead of weights and than retraining the network for reducing number of parameters. Unfortunately, these aforementioned models are not suitable for real world mobile devices because: 1) since these models enormously rises the computational operations, 2) the number of parameters is unarguably a substantial issue in application space which increases the memory cost due to large size of trained models on the disk.

Biological studies lead to the idea of Image pyramids (IP's) [131]. IP's shown to be an efficient data and processing structure for digital images in a variety of vision applications e.g. object, digit recognition [131]. IP's as well as NN are massively parallel in structure. Pyramids at their simplest are like stack of filtered images with exponentially reduced dimensions. Pyramids have a long relation with *ML* and *CV*.

Several models have been proposed based on the concept of pyramids e.g. Neocognitron, early LENET, Pyramidal neural network,

spatial pyramids, and several others [132, 14, 36] being discussed in Sec. 2.2. In some recent works, pyramid structure is used only in one layer like the Spatial Pyramid Matching (SPM) [36] for pooling layer. H. Fan et al. [37] tried to use pyramid structure in its last Conv layer of CNN for face recognition on LFW and showed 97.5% accuracy. P. Wang et al. [38] utilizes pyramid structure better than the aforementioned work. They applied temporal pyramid pooling to enhance and use the temporal structure of videos just like spatial pyramids in [133] where, they incorporated weak spatial information of local features. Their pyramidal temporal pooling method showed better results than the state-of-the-art two-stream model [39] on HMDB1 dataset. On the other hand, models like *PyraNet* [14], *I-PyraNet* [87], *LIPNET* [88] and their extended models emphasized strict pyramidal structure from input till output. Their objective is to show that following strict pyramidal structure can enhance performance as compared to unrestricted models.

We have explored and proposed some basic hints about the main questions regarding *CNN* for image classification problem. The questions we have identified to answer are: impact of reversing number of kernels in a convolutional layer?, does reversing a model work in every case?, impact of strict pyramidal structure?, how to reduce number of parameters without complex rules and loss of accuracy?. To answer these questions we have utilized some well-known state-of-the-art models e.g. LENET [132], AlexNet and its modified referenced model BVLC_Reference_Caffe [12]. We have shown that the same number of filters can be used in different but pyramidal order without affecting the performance of the base network.

5.2 Background

There are four key ideas behind *CNN's* that take advantage of the properties of an image: local connections, shared weights, pooling and use of many layers. Deep models or specifically *CNN* exploit

the compositional hierarchies of the images; in which higher-level features are obtained by composing lower-level ones. For example, local combinations of edges form motifs, motifs assemble into parts, and parts form objects. The role of the convolutional layer is to detect local conjunctions of features from the previous layer i.e. edges, axons, etc.; whereas, pooling layers not only reduces dimensionality, but also merges semantically similar features into one. This helps in providing translation and scale invariance to small shifts and distortions. Y. Lecun's [132] early CNN followed strict pyramidal approach. In addition, its weight sharing concept helped neural networks to reduce burden of large amount of trainable parameters. However, recent *CNN's* doesn't follow strict structure, although they reduced feature map size at each higher layer but increases number of maps as well resulting in further increase in total trainable parameters. Some of the well known models such as; *AlexNet* [12] that won ILSVRC had 60M parameters, DeepFace model with 120M parameters resulted in best face recognition accuracy, or a recent very deep model [126] that contains 133-144M parameters for getting about 24-30% top-1 error on ILSVRC-12 dataset starts from 96 feature maps and ends at 512 or more maps at a higher layer.

Therefore, some recent works like [134] and [96] highlighted the issue of memory usage in deep networks by reducing number of parameters. M. Lin et al. [96] proposed a unlikelier local patch modeling in *CNN* by replacing linear convolutions in each layer with convolving the input with a micro-network filter. This micro filter works like a multilayer perceptron. This technique is extended and used in inception model [31] as a micro-network modules. It works as dimensionality reduction to remove computational bottlenecks and reduce storage costs. Collins and Kohli [134] used sparsity-inducing regularizers during training to encourage zero-weight connections in the *Conv* and *FC* layers. T. Sanath et al. [135] exploited low-rank matrix factorization to reduce network parameters in FC layers. M. Denil et al. [136] tried to predict parameters from other parameters.

Strict pyramidal models [14, 137, 138] takes large amount of data as an input, refine it layer by layer in-order to reduce unwanted features and to enhance the final decision based on most likely and less number of unambiguous features. This gave base to many feature extraction and selection process in *CV* and *ML*. In *CV, SPM* with *SIFT+FV* technique dominated ILSVRC classification and detection competitions until *AlexNet* arrived [12]. Lazebnik et al. [36] introduces spatial pyramids technique which are mostly used for object recognition before deep *CNNs*. J. Masci et al. [139] used multi-scale pyramid pooling layer to get a fixed size input feature vector. Recently, a much deeper model is introduced having inspiration from both [36] and [139]. This spatial pyramid pooling *(SPP)* [129] approach provides multiple fixed size inputs to *FC* layer with the help of pyramid pooling, and shows better results than normal fixed size input models, e.g. Overfeat, AlexNet etc.

5.3 Proposed Model

Despite success of *CNN*, there is no principled way of finding good performing *CNN* architectures other than naive exploration of the architecture space. This is a question that many newcomers ask: how to arrange filters in each layer for modeling a good network? Less reasoning has been given other than; increasing number of maps as the network go deeper will give good results. Therefore, we have introduced simple rule of thumb when designing a *CNN* architecture based on the study of pyramidal neurons in the brain and human nature, highlighting this main question.

"If a deep CNN architecture with specific number of layers and filters works on a specific task, while keeping the number of layers same and reversing the number of filters, provided that it forms pyramidal structure, the resultant pyramidal architecture will result in same or better performance?"

This question highlights two main aspects. We ask if a network

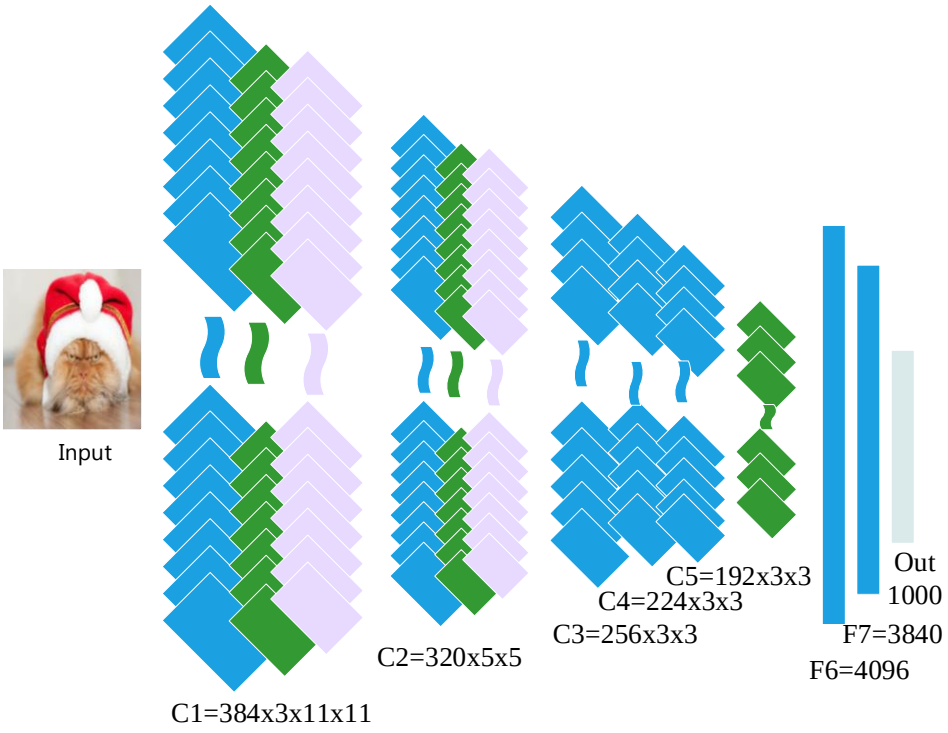
still works, if reversed, and, secondly, does it works only with a pyramidal structure. In addition to pyramidal structure, hints about this question is taken from a trick in PCA approach, where only changing the order of the matrices in calculating covariance matrix, not only reduce the time complexity but also avoided memory overflow issue [75]. Answering this question thus helped in proposing a general rule of modeling a network by reducing a constant factor $'f'$ of filters from layer $'l'$ compare to previous layer $'l-1'$, other than better understanding the network behavior.

Based on this reduction factor $'f'$ of filters from the previous layer we have investigated a strict pyramid network *SPyr_CNN* architecture as shown in Fig. 5.1. It starts from a big input/first layer and then refining the features at each higher layer, until it achieves a reduced most discriminative set of features as being done in [14, 87, 88]. Imposing strict pyramidal structure should not only retain and improve accuracy, but also retain/improve accuracy and reduce the number of parameters which results in less number of memory space on the disk. And this makes *CNN's* more feasible for applications where there is lack of memory. To achieve this goal, we will present experimental evidences based on comparison with state-of-the-art models and datasets in the following section.

5.4 Experimental Results

Our strictly pyramidal CNN architectures are evaluated in two ways. First type of models provide empirical evidence in sub-sections 5.4.2, 5.4.3, 5.4.4, and 5.4.5 that a *SPyr_CNN* on small to large benchmark datasets *MNIST, CIFAR-10, CIFAR-100* and *ImageNet-12* performs similarly or better than complex state-of-the-art deep CNN models despite reduction in parameters. Whereas, in the second type we used a version of Siamese network for person re-identification problem applied on *VIPER* dataset in sub-section 5.4.6. We showed that the strict pyramid structure imposed on Siamese network re-

Figure 5.1: Strictly Pyramidal Architecture for CNN (SPyr_CNN).



duces ambiguity and enhances Rank-1 performance in identification of a person, crucial for surveillance applications.

5.4.1 Datasets

5.4.1.1 MNIST

MNIST is a well-known 10 class hand written digits recognition dataset [140]. It consists of 60000 training and 10000 testing images. Each class have 5000 training images and 1000 testing images.

5.4.1.2 CIFAR-10

Cifar-10 is an object recognition dataset [141]. It consists of 60000 colour images with 6000 images per class. In each class 5000 images are used for training where as remaining 1000 images are used for

testing. Similar to *MNIST*, each class have 5000 training images and 1000 testing images. We have used these datasets to perform an in depth study of different models based on comparison with state-of-the-art, since the CNN can be easily trained and tested with reasonable computing and time cost.

5.4.1.3 CIFAR-100

Cifar-100 is also an object recognition dataset having 100 classes but same number of images as Cifar-10 [141]. However, due to less images per class that results in less training data, it is considered medium size challenging dataset.

5.4.1.4 ImageNet-12

ImageNet-2012 dataset consist of 1000-classes [29]. It comprises of 1.2 million training images and 50,000 validation images (used as testing set). The results are measured by top- 1 and top-5 error rates. We have only used the provided data for training; all the results are evaluated on the validation set. Top-5 error rate is the metric officially used to rank the methods in the classification challenge. We perform two different experiments in order to evaluate power of pyramidal networks, i.e. with full training sets and with reduced training sets. Hence, we reduce 10% and 20% randomly selected images from Cifar-100 and ILSVRC-12 training dataset. To generalize a common rule of selecting number of filters in each layer of a deep network, we use Cifar-10 and ILSVRC-12 for examining the best decrement factor for a successful model. These models are explored in the up-coming sections.

Strict pyramidal models will be represented by *SPyr* in beginning of each model. We have implemented these *SPyr_CNN* models on widely deployed Caffe [58] framework. The hyper-parameters for training setup are all same as *AlexNet* model and its variant BVLC_Refference_Caffe (*BVLC_Ref*) model in Caffe [58] i.e. weight

decay as 0.0005, momentum as 0.9, and dropout of 0.5 is used in FC layers. In pre-processing nothing sophisticated has been done. We only used off-the-shelf normalization method available with Caffe. However, the learning parameters vary according to the dataset and respective state-of-the-art method being used in literature. To calculate number of parameters for each layer we adopted number of parameters as $R1 \times R2 \times C \times M$, where $R1$ and $R2$ are rows and column size of kernel, C is number of channels in that layer, and M represents number of kernels for upper layer that will results in M output feature maps.

5.4.1.5 VIPeR

VIPeR dataset [142] is a person re-identification benchmark dataset. We used it in our experiments by dividing it in training and testing set, similar to the work being done in [142, 143, 144]. It contains 632 target subjects, each having 2 images from 2 different camera views (camera A and camera B). We took 11 sets by randomly selecting 316 disjoint subjects for training and remaining 316 for testing set. The first split called Dev. view or training set is used selecting optimal parameters while the other 10 splits called Test view are used for reporting the results. This dataset is used in Sec. 5.4.6 to show that our model reduces ambiguity and enhances Rank-1 performance.

5.4.2 Impact of Pyramidal Structure

Few standard *CNN* architectures are examined for the impact of reversing and imposing a pyramidal structure to the networks. The optimal result for *MNIST* and *Cifar*-10 is shown respectively by *Caffe_LENET* and *C10* in Tab. 5.1. *Caffe_LENET* has 20 filters in first convolutional layer (C1) and 50 filters in second convolutional layer (C2). If we reverse it, i.e. $C1 = 50$ and $C2 = 20$, we call it *SPyr_Rev_LENET* and it resulted in competitive accuracy as shown in Tab. 5.1. Same is the case for Cifar-10 i.e. *SPyr_Rev_C10*. Cifar-

100 consisted of 50 images per class consisting of 50000 training images. This makes it really challenging and we faced a 0.37% loss in overall accuracy as shown in Tab. 5.1. *SPyr_Rev_C100*.

BVLC_Ref is similar to *AlexNet* but with a slight change in order of pooling and normalization layer [58]. *BVLC_Ref* model in reverse does not provide a pyramidal structure i.e. *Rev_BVLC_Ref* has 256, 384, 384, 256, and 96 kernels at C1, C2, C3, C4, and C5, respectively. As expected, it does not learn. Therefore, we rearranged it, not only with the aim to provide a pyramidal structure but also maintaining the same number of filters. Hence, Tab. 5.1. *SPyr_Rev_BVLC* gets 352 kernels in C1 and C2, followed by 256 kernels in C3 and C4. Finally, C5 layer gets 160 kernels due to adding 32 kernels from C1 and C2. As shown, reverse *SPyr* models helped each standard *CNN* model in increasing and retaining overall performance. However, in big networks we learned one important point: if the reverse of a model doesn't result in pyramidal structure, the performance will drop or the network will not learn at all; this is true in the case of *Rev_BVLC_Ref* model.

The question of how many kernels one should have at first layer or at each layer is still not fully theoretically asserted and is an open research problem. However, in our experiments with *SPyr_CNN's*, we experimented that the size of last 'conv' (C5) layer has high impact on overall accuracy. Therefore, we concluded that there should be enough number of filters at first layer (C1), so that, after reduction in several layers, the last convolutional layer could get almost 40-60% of the filters in C1. Otherwise, there would be a 1-3% increase in error rate, or in worse case, it will not learn at all, e.g. *Rev_BVLC_Ref*.

This is achieved reducing the number of kernels by a factor $'f' = 10, 15, 20\%$ from each layer to the next one, as shown in Tab. 5.2. This ensures pure pyramid structure unlike *SPyr_Rev_BVLC* where we had two layers having the same number of kernels. We also tested the model on *Cifar-10* and *ImageNet*. *Cifar-10* shows that with 10-15% reductions in kernel number gets almost the same performance.

Whereas, with a 20% reduction, accuracy decreases by 0.86%. In big models like ImageNet, our *SPyr_BVLC_Ref*** with 10% reduction in each layer including FC's layers improves accuracy by 0.59% and 0.61% compared to our *SPyr_BVLC_Ref* and *BVLC_Ref*, respectively (see Tab. 5.2). Whereas, models with 20% reduction got bad results or did not learn due to few kernels in 'C5'. Further, *BVLC_Ref* [58] results in maximum accuracy at 313000 iteration, whereas, we reported at 393000. However, at the end of 450000 iterations our *SPyr_BVLC_Ref*** achieved 0.61% accuracy.

5.4.3 Parameter Reduction & Size on Disk

Another big impact of *SPyr* approach is its reduction of parameters. It results in less size of trained model on disk as can be seen in Tab. 5.3. We did a number of experiments with *MNIST* to see how much we can reduce parameters. At first glance, *SPyr_Rev_LENET* reduced 55.5% parameters and enhanced performance by 0.02%. This reduction in parameters resulted in less space on disk i.e. with $431K$ parameters it needs $1.685MB$ whereas, later it took 0.749MB space on disk. We examined it further and designed *SPyr_LENET* with 80% reduction in parameters. It resulted in 0.03% enhancement over *Caffe_LENET*. However, when we reduced more than 90% of parameters from convolutional and FC layers, it retained same accuracy as shown in Tab. 5.3. *SPyr_LENET**. *SPyr_LENET*** shows that if we concern about accuracy instead of memory space, than *SPyr* models can perform much better. Similarly for Cifar-10 and Cifar-100, reverse models in Tab. 5.3 not only reduces parameters but also gave competitive result.

In case of ImageNet, the size of the *BVLC_Ref* trained model is $238MB$. Whereas, our *SPyr_BVLC_Ref* and *SPyr_Rev_BVLC* have $198MB$ for better results and $160MB$ for 1% less accuracy, respectively. This is due to proper model selection having $10-20$ million fewer parameters resulting in reduction of ambiguity. Reduction is not only in 'Conv' layers but also from 'FC' due to less maps con-

Table 5.1: Results for Referenced Models vs. their reversed model according to our statement

MNIST

Caffe_LENET = 20-50-500-10

SPyr_Rev_LENET = 50-20-500-10

Model	Train Loss	Test Loss	Accuracy
Caffe_LENET	0.003735	0.029231	99.1
SPyr_Rev_LENET	0.003547	0.025142	99.13

CIFAR-10

C10 = 32-32-64-10

SPyr_Rev_C10 = 64-32-32-10

Model	Train Loss	Test Loss	Accuracy
C10	0.331029	0.532539	81.65
SPyr_Rev_C10	0.321514	0.537438	81.67

CIFAR-100

C100 = 150-170-200-100

SPyr_Rev_C100 = 200-170-150-100

Model	Train Loss	Test Loss	Accuracy
C100	0.53676	1.5585	58.64
SPyr_Rev_C100	0.63625	1.5549	58.27

ILSVRC-12

BVLC_Ref=96-256-384-384-256-4096-4096

Rev_BVLC_Ref=256-384-384-256-96-4096-4096

SPyr_Rev_BVLC=352-352-256-256-160-4096-4096

Model	Train Loss	Test Loss	Accuracy
BVLC_Ref	1.21952	1.840	57.03
Rev_BVLC_Ref	Does	Not	Learn
SPyr_Rev_BVLC	1.34997	1.842	56.81

Table 5.2: Reduction in Kernels by Factor $'f'$ along with their accuracies

Cifar-10

Model	$'f'$	Params	Accuracy %
(A) 64-58-52-10	10	245306	83.43
(B) 64-54-44-10	15	197111	83.36
(C) 64-52-40-10	20	158623	82.57

ILSVRC-12

(D) SPyr_BVLC_Ref**= 384-346-308-270-232-4096-3687-1000

(E) 384-346-308-270-232-4096-4096-1000

(F) 384-326-270-212-156-4096-3687-1000

(G) 384-326-270-212-156-4096-4096-1000

(H) 384-308-232-156-80-4096-3277-1000

Model	$'f'$	Params	top-1%	top-5%
(D)	10	58M	57.64	80.68
(E)	10	60M	57.62	80.64
(F)	15	46M	56.38	79.67
(G)	15	48M	56.61	79.76
(H)	20	33M	-	-

nected to neurons. These results showed that this approach can provide significant difference in specific real world application, where lack of memory storage is an issue.

S. Han et al. [30] reduced parameters with a time consuming three step process. Rather than training weights, they learn those connections which are more important at first step, followed by pruning those connections, and retraining the remaining connection and their weights at final step. They achieved 99.23% accuracy with $12x$ reduction in parameters but consumed $2x$ more time than normal, as shown in Tab. 5.3 Caffe_LENET. In addition, they applied complex regularization and loss functions to find the optimal results. However, *SPyr_LENET*** achieved 99.24% accuracy with about 50% reduction in parameters and almost same amount of time (i.e. $47.6s \pm 1$ on NVIDIA Quadro K4200 GPU) as *Caffe_LENET* (other simulations were also running on same computer) as shown in Tab. 5.3. Further, our reduction is not only done in 'FC' layers

Table 5.3: Parameters and their size on disk for base and pyramidal architectures along with their accuracies

MNIST

(A) Caffe_LENET = 20-50-500-10

(B) SPyr_Rev_LENET = 50-20-500-10

(C) SPyr_LENET = 35-15-500-10

(D) SPyr_LENET* = 25-15-100-10

(E) SPyr_LENET** = 100-68-100-10

Model	Parameters	Size in MB	Accuracy
(A)	431080	1.685MB	99.1
(B)	191830	0.749MB	99.13
(C)	**48910**	**0.191MB**	**99.14**
(D)	**35150**	**0.137MB**	**99.1**
(E)	**219250**	**0.857MB**	**99.24**

CIFAR-10

(F) C10 = 32-32-64-10

(G) SPyr_Rev_C10 = 64-32-32-10

(H) SPyr_C10 = 38-30-24-10

(I) SPyr_C10 = 84-64-44-10

Model	Parameters	Size in MB	Accuracy
(F)	87978	0.335MB	81.65
(G)	**83658**	**0.319MB**	**81.67**
(H)	51392	0.189MB	80.9
(I)	214142	0.837MB	83.04

CIFAR-100

(J) C100 = 150-170-200-100

(K) SPyr_Rev_C100 = 200-170-150-100

(L) SPyr_C100 = 200-128-64-100

Model	Parameters	Size in MB	Accuracy
(J)	1811870	6.917 MB	58.64
(K)	1733120	6.616MB	58.27
(L)	952692	3.637MB	57.23

ILSVRC-12

(M) BVLC_Ref [58]=96-256-384-384-256-4096-4096

(N) SPyr_BVLC_Ref=384-320-256-224-192-4096-3840

(O) SPyr_Rev_BVLC=352-352-256-256-160-4096-4096

(P) SPyr_BVLC_Ref*=352-352-256-256-160-3840-2840

(Q) SPyr_BVLC_Ref**=384-346-308-270-232-4096-3687-1000

Model	Parameters	Size in MB	Accuracy
(M)	62378344	238.15MB	57.03
(N)	**54046920**	**198.26MB**	**57.05**
(O)	43065448	160.93MB	56.81
(P)	40867904	151.71MB	56.40
(Q)	58741371	224.2MB	**57.64**

Table 5.4: Best Strictly Pyramidal Models and their Accuracies
(A) MNIST_Caffe_LENET = 20-50-500-10
(B) SPyr_C10 = 52-42-32-10
(C) SPyr_Rev_C100 = 200-170-150-100
(D) SPyr_BVLC_Ref=384-320-256-224-192-4096-3840

Model	Parameters	Train Loss	Accuracy
(A)	48910	0.00994945	99.14
(B)	94756	0.319729	81.94
(C)	1733120	0.445774	58.27
(D)	54046920	1.24036	57.05

but also in 'Conv' layers unlike the *LeNet-5 Pruned* Tab. 5.3. (F) [30] which reduced parameters mainly in 'FC' layer. *Alexnet_Pruned* [30] achieved same result with $6.7M$ parameters, but their model took about $700K$ to $1000K$ iterations. Whereas, we achieved same results with simple techniques, $10M$ reduction in parameters and same number of $450K$ iterations. However, due to more maps in initial layer, our model takes slightly more time than *BVLC_Ref.* In Fig. 5.3, D100_Model_1 shows one of our pyramidal models that reduced 30.9623% parameters with only 0.0142% degradation in performance compared to reference model. Tab. 5.4 summarizes the best models in-terms of parameters and accuracy for *MNIST, Cifar-10, Cifar-100* and *ILSVRC-12.*

5.4.4 Performance of Pyramidal Models with Less data

One of the questions raised against *CNN* is that it doesn't work if small datasets are available. Therefore, we tested *SPyr_CNN's* with reduced data. We divided Cifar-100 and ILSVRC-12 in two new random training sets, i.e. 90% and 80% of the original one. Despite reduction of $150K$ and $300K$ randomly selected images for test1 and test2, respectively, we got only 1% gradual degradation in overall accuracy (see Tab. 5.5). So, if we properly understand *CNNs*, we can model such architectures in order to provide optimal results even with small training datasets. Our top-1 accuracy even with

Table 5.5: Evaluating generalization power of Strictly Pyramidal networks with reduction of training Data medium and large datasets. top-1 and top-5 represents error.

SPyr_C100 = 200-170-128-100

Data	Train Loss	Test Loss	top-1	top-5
100%	0.636249	1.57851	41.93	-
90%	0.615743	1.64351	42.66	-
80%	0.407365	1.72381	44.14	-

SPyr_BVLC_Ref=384-320-256-224-192-4096-3840

Data	Train Loss	Test Loss	top-1	top-5
100%	0.449904	1.55493	42.9	19.7
90%	1.32812	1.88455	44.0	20.7
80%	1.25612	1.95334	45.1	21.4

90% data is better than reported in [134] by 0.4%. However, when we reduced data other than 10%, the performance drops down below 0.7%.

Performance of our *SPyr_CNN_C100* with reduced data while training is shown in Fig. 5.2 (with 100%, 90% and 80% training data). Not only the model with 100% training data, but also the models with 90% and 80% data learn smoothly. Similarly, to see the training behavior of ILSVRC-12 with reduced data, Fig. 5.4 shows its learning behavior during training. Whereas, their performance through out the training process is shown in Fig. 5.3. The behavior for *SPyr_BVLC_Ref* with 100% (D100_percent), 90% (D90_percent) and 80% (D80_percent) is almost smooth without any big falls or degradation. Curves shows almost similar nature despite less training data.

5.4.5 Comparison with State-of-the-art

We compared the pyramidal structure mainly with Caffe reference models i.e. *AlexNet* and *BVLC_Ref* as well as some state-of-the-art models as can be seen in Tab. 5.6. Our *SPyr_LENET* model for MNIST outperformed based model as discussed in Sec. 5.4.3. We

Figure 5.2: Performance evaluation based on Accuracy for CIFAR-100 with reduction in training data for total of 70000 iterations

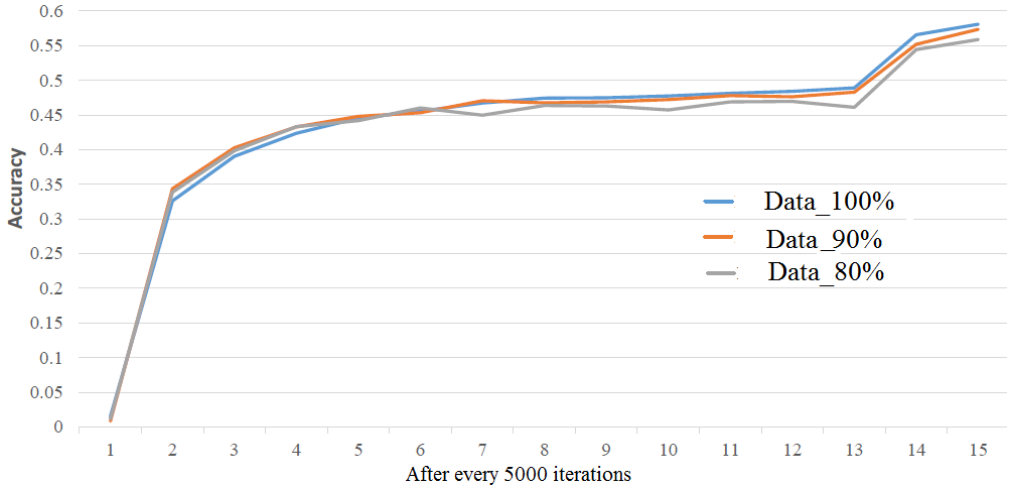
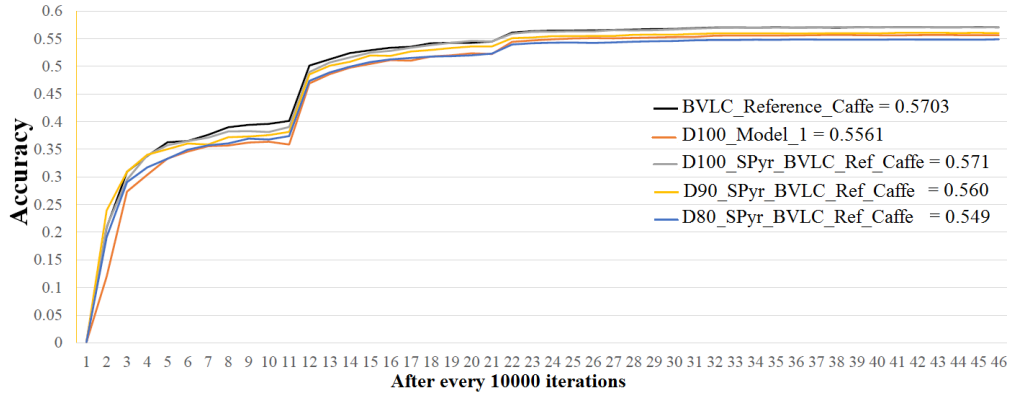
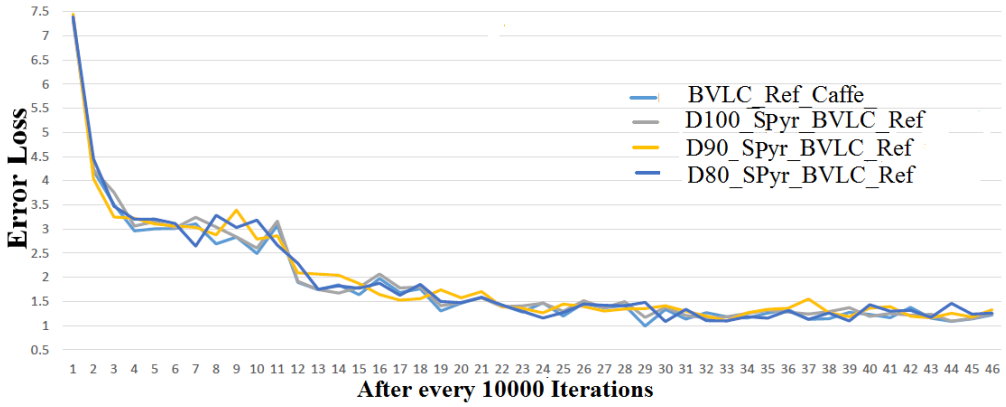


Figure 5.3: Comparison of Validation Accuracy with less data after each 10000 iterations for total of 450000 iterations



achieved comparable results in case of Cifar-10 and Cifar-100 from the base models, i.e. 81.65% vs 83.34% or 58.64% vs 58.27%. However, in comparison to models like [145, 96], our results are $5-8\%$ less accurate. By observing their model, unlike ours, they contain a million parameters as well as results from sophisticated and complex models whereas, we achieved with following our assumption of reversing and imposing a *SPyr* structure. Hence, if we neglect memory and increase parameters by increasing the *SPyr* network, the performance improves. In case of large scale dataset, the results are quite

Figure 5.4: Training Loss with 10000 iterations difference for total of 450000 iterations



promising in terms of accuracy. However, it took slightly more time due to more maps at initial layers. This limitation can be avoided by selecting a proper model. As *SPyr_LENET** shows that *SPyr* models can provide better or competitive results even with smaller number of kernels. In terms of reduction of parameters, recent models i.e. Tab. 5.6. (F) [134] and (G) [30] reduce far more parameters than our models Tab. 5.6. (C) and (D), but they achieve less top-1 and top-5 error for ILSVRC-12.

Some researchers are trying to avoid fully connected layers as majority of parameters are from those layers. Though, it should be noticed that these layers have great impact on over all accuracy. These are highly dependent on 2D or 3D layers. Therefore, one of the solution is to reduce the size of the last convolutional layer as-well-as number of neurons in specific order. Pyramidal structure is quite feasible for this scenario. We used pyramid structure in *FC* layers as can be seen in Tab. 5.3. (N), (O), and (Q). Tab. 5.3 (N) shows better result by following *SPyr* structure while Tab. 5.3 (O) fallback with only 0.63%, whereas, Tab. 5.3 (Q) gives the best result by reducing kernels and neurons at a constant factor of 10. To visualize and understand the output of our trained model, we have shown output maps resulted by first layer of trained *BVLC_Ref* model and our *SPyr_BVLC_Ref* in Fig. 5.5 and 5.6, respectively. The maps pro-

Table 5.6: Comparison with Stat-of-the-art in-terms of Error% and Less number of Parameters

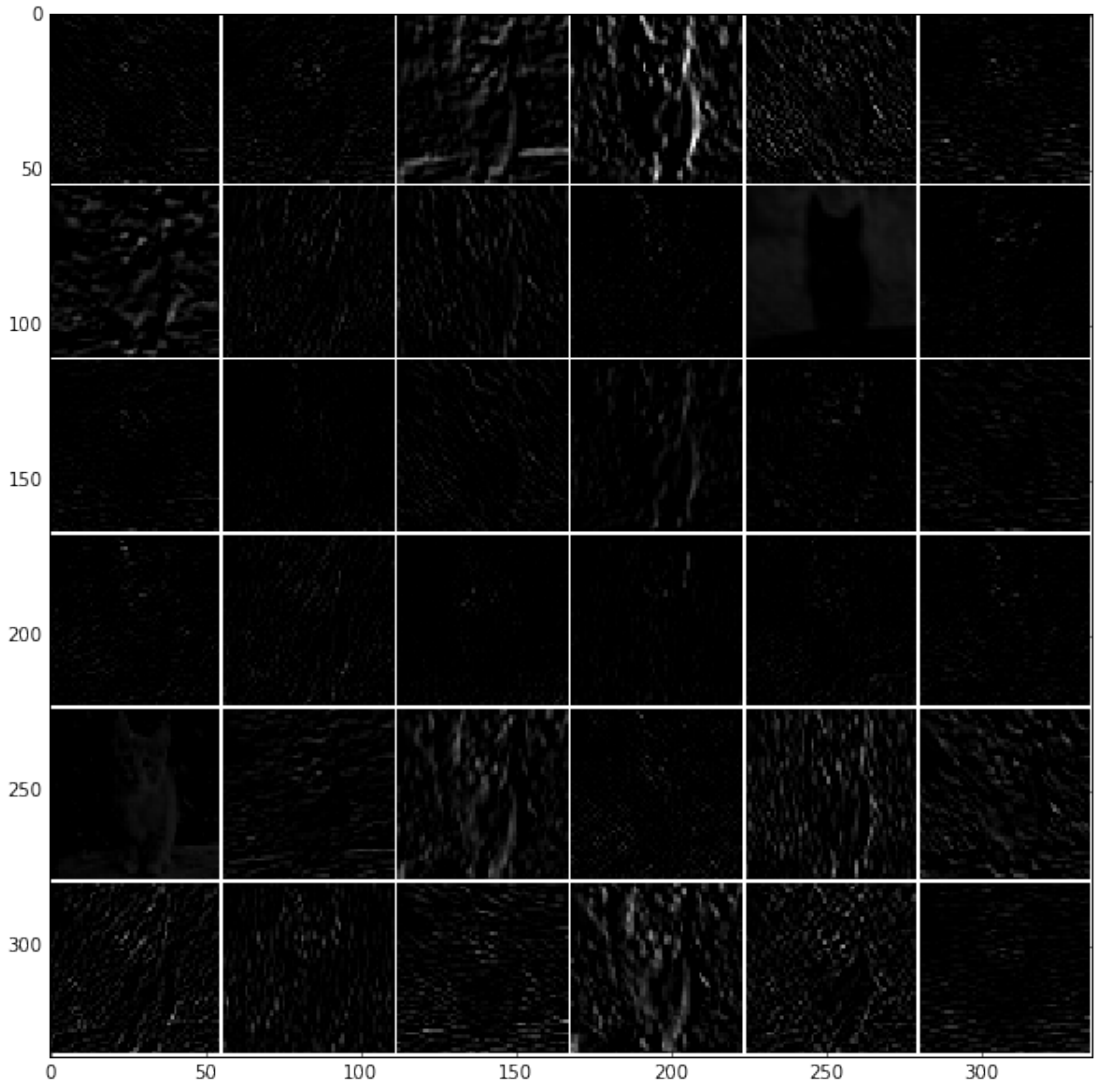
Model Cifar-100	Error %
C100	44.49
Stochastic_Pooling [133]	42.51
SPyr_C100	41.73
NIN [96]	35.68
Deeply Supervised Networks [145]	34.57

ILSVRC-12
(A) Alexnet [58]
(B) BVLC_Ref [58]
(C) SPyr_BVLC_Ref**
(D) SPyr_BVLC_Ref
(E) SPyr_BVLC_Ref*
(F) Sparse_MemoryBounded [134]
(G) Alexnet_Pruned [30]

Model	Params	top-1 %	top-5%
(A)	60M	42.9	19.8
(B)	60M	42.6	19.6
(C)	58M	42.3	**19.3**
(D)	50M	42.9	19.7
(E)	40M	43.2	19.9
(F)	15M	44.4	19.6
(G)	7M	42.8	19.7

duced by our model are more clear, smooth and understandable as compare to the maps produced by referenced model. This shows that how better and fine grained information is extracted from real input image. Still, a more detailed analysis is desired to precisely assess the effect of *SPyr* architectures on new deep models and *ImageNet* datasets. Such a comprehensive quantitative study using multiple networks is extremely time demanding and it is in our focus for future work.

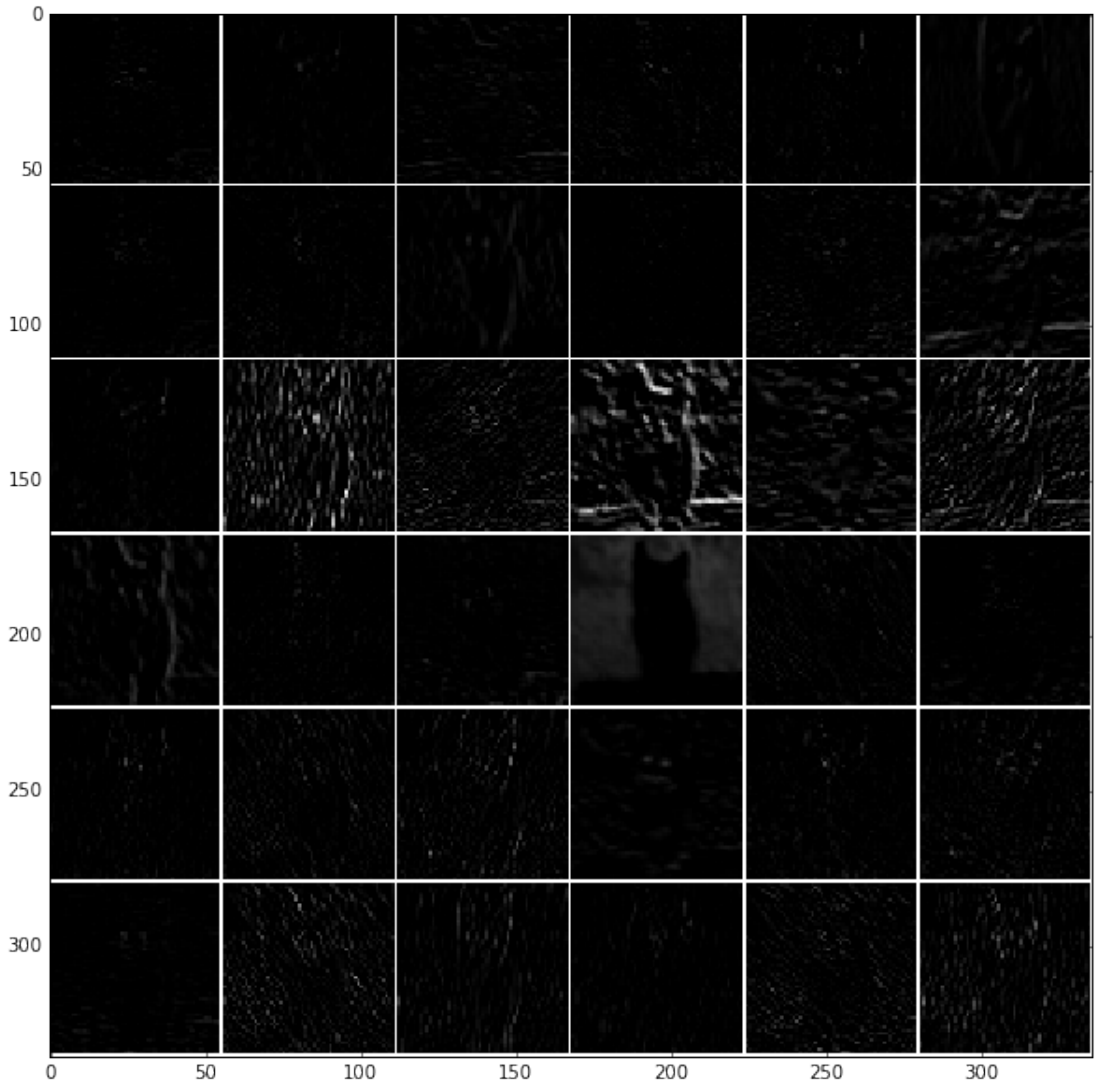
Figure 5.5: Output maps of first convolutional layer of Caffe trained model



5.4.6 Reducing Ambiguity

Person re-identification (PRe-ID) is an open and recently introduced problem in CV trying to answer questions like: "Have I seen this person before? or Is this the same person?". Formally, it is about identifying an individual in varying locations, views, and lighting conditions over a set of non-overlapping camera views in a small resolution images, e.g. 48×128 pixels. These factors make it very hard

Figure 5.6: Output maps of 1^{st} convolutional layer of our SPyr_BVLC_Ref trained model



and an important task of *CV* with applications ranging in many contexts from intelligent video surveillance, people tracking and behavior analysis due to its ability to recognize a subject without its physical interaction with the system.

Cumulative matching characteristic (CMC) curve [146, 147] is a commonly used performance measure in evaluating PRe-ID. The CMC curve represents results of an identification task by plotting

the probability of correct identification (y-axis) against the number of candidates returned (x-axis). The faster the CMC curve approaches one, the better is the PRe-ID algorithm. In a PRe-ID, a better rank-1 recognition rate is often preferred [148]. Therefore, following [148] our aim is also to improve the Rank-1 recognition rate among the k best candidates, e.g., $k < 20$, crucial for many real-world surveillance applications. In this work, all the experiments are done on *VIPeR* dataset explained in Sec. 5.4.1.5.

Recently, Improved Deep Metric Learning (Improved-DML) model [144] combines feature extraction and metric learning in a unified framework, like the Siamese *CNN*. An image is divided into three parts (head, abdomen, and foot region) that is given as input to Improved-DML. We enhanced this model and propose two variants of this model, i.e. One with strict pyramid structure (SP-Improved-DML) and the other without pyramid structure (Non-SP-Improved-DML). The *SP-Improved-DML* results in a strictly pyramidal Siamese *CNN* that enhances the Rank-1 performance in all the cases having fewer number of parameters than the non pyramidal model.

5.4.6.1 SP-Improved-DML

A Siamese architecture takes an image pair as input and the output is binary value, revealing if the two images comes from the same subject, or not. The weights between the two sub-networks in Siamese architecture can be shared or not. For person re-identification problem, the best choice is clearly sharing parameters, to find out the peculiarity of an individual in different pose, or from images acquired by different views. Our architecture, is composed by two siamese strict pyramidal *CNN* blocks, B1 and B2, a connection function, C and a cost function, J. Driven by a common objective function, this model can learn, simultaneously, both the features and an optimal metric to compare them. The strictly pyramidal CNN block is composed by 2 convolutional layers, C1 and C3.

The model imposes strict pyramidal structure on the the Improved-

DML (SP-Improved-DML) for enhancing Rank1 performance. In first phase we use a *CNN* as shown in Fig. 5.8 which is composed by 2 convolutional layers, 2 max pooling layers and a full connected layer. This is similar to the one used by Improved-DML, however the difference among the two is their input size, structure and batch arrangement. This differences consist of:

- not using cross-channel normalization unit.
- using full image picture rather dividing image in three parts and giving as input to three parallel *CNN's*.
- using shared parameters in both sub-networks rather non shared which is good for general PRe-ID problem where the change is in the pose.
- using simple hyperbolic tangent function instead of ReLU [12] as activation function for each layer.
- Imposing our strict pyramid structure that reduce ambiguous features.

Our Non-SP-Improved-DML has all the above differences compare to the base Improved-DML model other than not imposing pyramid structure in the model. It will show the role of pyramid structure in increasing rank-1 performance and reducing the number of parameters. The SP-Improved-DML is shown in Fig. 5.7, the number of kernels in convolutional and pooling layer shows the difference. The pyramid structure starts with more filters/kernels in first layer (C1), i.e. 32 having kernel size of 7×7. It is followed by a pooling layer that further reduces its spatial dimension and than another convolution layer (C3) with 25 kernels of size 5×5. In the end, it contains a fully connected layer that outputs a vector of 500 dimensions. This pyramid structure not only extract several type of features from the real input, but it refined them as it goes deeper and deeper by reducing redundant and ambiguous features. Finally, it provides fewer

number of features but highly discriminative for classification purpose. This structure reduced the number of features by more than 50% and provided better rank-1 performance on almost all ranks.

An important factor in this model is the use of connection function. High level features of all parts are fused at F5 layer by sum rule. Then, the similarity of fused features are evaluated by the connection function (5.1) as used in [149].

$$S_{cos}(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i x_i}\sqrt{\sum_i y_i y_i}} \tag{5.1}$$

Here *cos* represent cosine function, x, y belongs to the feature vector of dimentsion 500 obtained by f5 layer of blocks, B1 and B2. The advantage of the cosine function is that its derivative is simple to be calculated; Furthermore, the similarity value is between -1 and $+1$, where -1 indicates not equal vectors and $+1$ indicates that the two vectors are equal. It is driven by a common cost function derived from binomial deviance, in which combined features contribute equally and jointly to the training process. The cost function is given below:

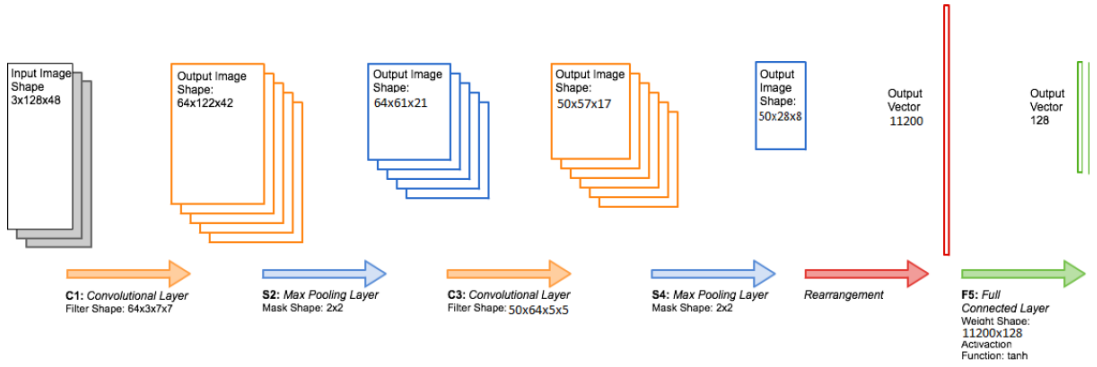
$$J_{dev} = \sum_{a}^{b_{i,j}} W \circ ln(e^{\alpha(S-\beta)\circ M} + 1) \tag{5.2}$$

Where $\circ$ is the dot product between the matrices, and $S_{i,j}$ is similarity between the two samples 'i' and 'j', M as mask matrix for 'i' and 'j', and W as weight of that connection, given as

$$S = [S_{i,j}]_{nxn}, \qquad where \qquad S_{i,j} = S(x_i, x_j), \tag{5.3}$$

$$M = [M_{i,j}]_{n\times n},\ where\ M_{i,j} = \begin{cases} 1 & for\ \ +ive\ \ pair \\ -c & for\ \ -ive\ \ pair \\ 0 & for\ \ neglected\ \ pair \end{cases} \tag{5.4}$$

Figure 5.7: Strictly Pyramidal CNN (SP-CNN)



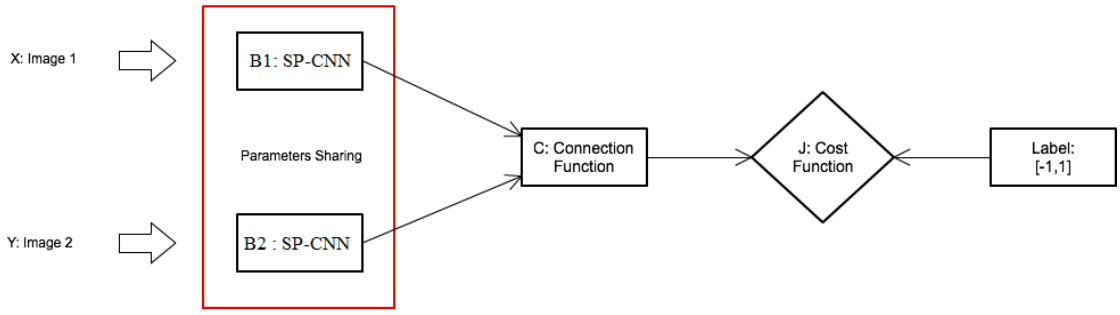
$$W = [W_{i,j}]_{n\times n},\ where\ W_{i,j} = \begin{cases} \frac{1}{n_1} & for\ \ +ive\ \ pair \\ \frac{1}{n_2} & for\ \ -ive\ \ pair \\ 0 & for\ \ neglected\ \ pair \end{cases} \quad (5.5)$$

Here $M_{i,j}$ denotes either they are the same subject or not. The number of positive and negative pairs are denoted by n_1 and n_2, respectively. These negative and positive pairs are deeply separated by minimizing Eq. 5.2. In the end, traditional mini-batch stochastic gradient decent is used to update the shared weight parameters during training phase. 'c' represents asymmetry cost on the label mask in $M_{i,j}$ on each positive and negative pairs. It regularize the network e.g. 1 for $+ive$ and 2 for $-ive$ pair. This regularization shows promising enhancement in overall performance of the network.

5.4.6.2 SP-Improved-DML Performance

To evaluate the performance, we used the same architecture as our Non-SP-Improved-DML. But used more filters at initial layers and reduced it gradually in higher layers. We tried to examine it in two scenarios. First to check it with same number of filters as used in Non-SP-Improved-DML i.e. 128 (64+64). We adopted three combinations i.e. (96,32), (80,48), and (70,58). Secondly we used a small combination of kernels, to see the power of this pyramid approach

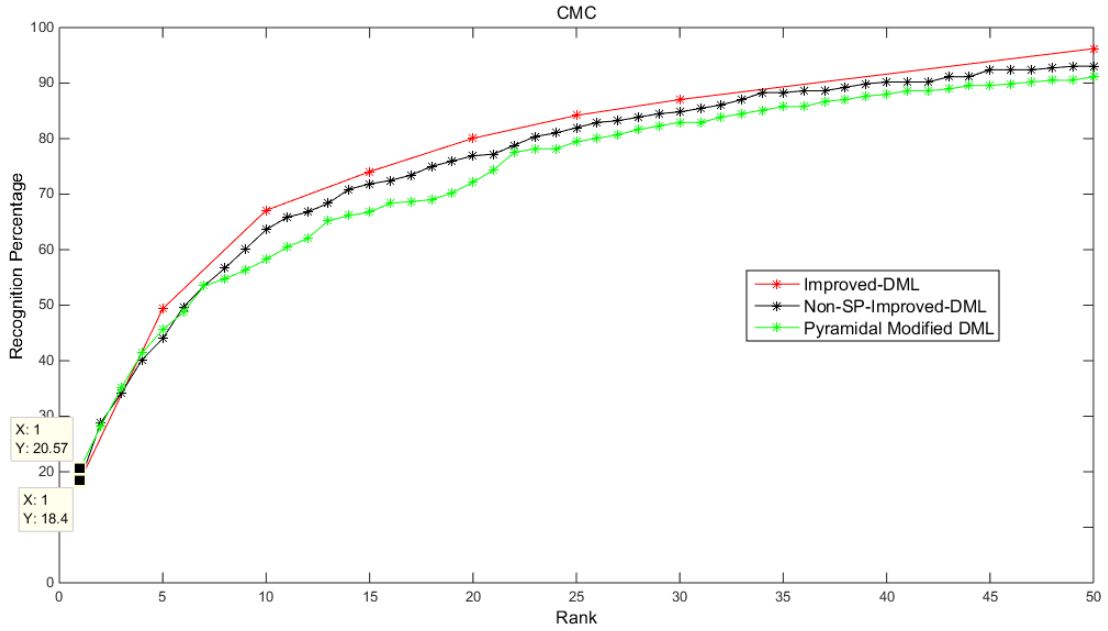
Figure 5.8: Structure of SP-Improved-DML



for which we have taken filters combination of only 32 and 25. This is more than 50% reduction of filters as compared to Improved-DML and Non-SP-Improved-DML. The results achieved with these set of filters on *VIPeR*are shown in Fig. 5.9 and Fig. 5.10. Fig. 5.9 shows the best optimal results interms of Rank-1 and least number of filters and parameters. In which SP-Improved-DML outperforms Improved-DML and Non-SP-Improved-DML by 2.17%. However, the performance after Rank-4 gradually decrease compare to other two methodologies. In real world scenarios, Rank-1 is more important and acceptable as many reported it with great emphasis in literature. It is necessary to establish the identity of a person in times as short as possible. In fact, the 'Rank-1 re-identification' require a computational time equal to $O(n)$, while the 're-identificaition' higher rank requires at least $O(nlogn)$. Our model shows an edge over traditional model in this regards by showing almost similar results with only 50% of the parameters compare to previously mentioned. The training time has been reduced from 3 hours and 31 minutes to 2 hours and 40 minutes.

Fig. 5.10 shows results for different pyramid structures. In all of them, one can notice that all sets of pyramid structures gives better results than stat-of-the-art for Rank-1 to Rank-4. However later the performance decreases gradually. In addition the Rank-1 further increases with increase in number of filters at C1 layer. The best Rank-1 with equal number of filters i.e. 128 gives more higher

Figure 5.9: CMC Rank-1 accuracy for Improved-DML, Non-SP-Improved-DML and SP-Improved-DML



rank, i.e. 20.2% increasing the rank by nearly 3%. This clarifies that pyramidal structure does refine the features, reduces ambiguity that provide higher Rank-1 results compare to other conventional non-strict pyramidal architectures.

5.5 Conclusion

We have demonstrated empirically that giving pyramidal structure to *CNN's* can lead a scale down in the number of parameters as well as less solver memory consumption on disk, hence producing competitive results. Our experimental analysis are carried out on five standard datasets, which showed the effectiveness of the pyramidal structure. Training with reduced training data shows similar and smooth learning with slight decrease in overall accuracy, i.e. about $0.5-1\%$ decrease with each 10% reduction. A suggestion for selecting number of kernels in each layer, especially first and last convolutional

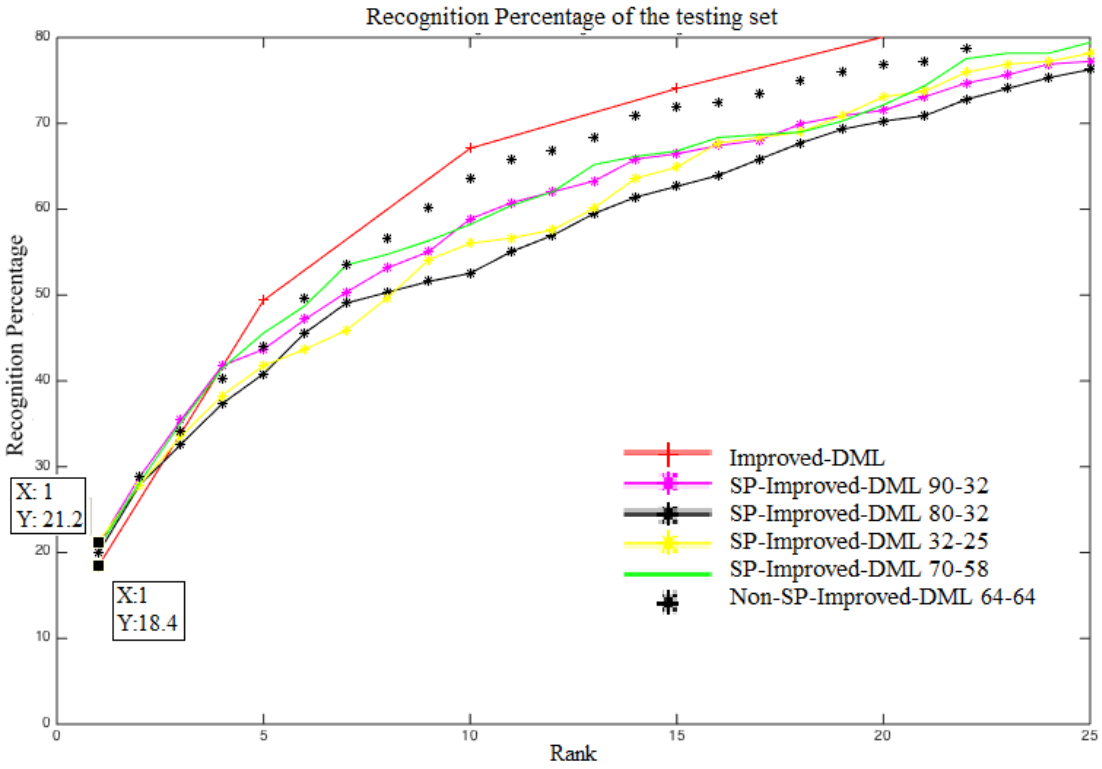


Figure 5.10: Rank-1 for SP-Improved-DML models by using different number of filters combinations

layer is given. In a sense, it makes it even more surprising that simple strict pyramidal model outperforms many existing sophisticated approaches. This pyramidal structure reduces ambiguity, hence increases Rank-1 performance for PRe-ID as well as other recognition problems.

5.6 Related Publications

- Ihsan Ullah and Alfredo Petrosino. *About Pyramid Structure in Convolutional Neural Networks*, in Proceedings of International Joint Conference on Neural Network (IJCNN-16), pages 236-245, 24-29 Jul, 2016.
- Sara Iodice, Alfredo Petrosino and Ihsan Ullah. *Strict Pyramidal Deep Architecture for Person Re-Identification*, in Book: Advances in Neural Networks, pp: 179-186, 2016.

Chapter 6

Conclusion

In first part of this thesis, our aim is to highlight pyramidal neural networks. We give intuition for following strictly biological plausible pyramidal structures of the decision making in human brain. It should rather be assumed as an attempt to search for a minimum necessary ingredient for designing a deep architecture. We present a simple idea of pyramidal structure based on internal structure of pyramidal cell in brain and a hint from PCA. A strict 3D pyramidal neural network and a strictly pyramidal *CNN* architecture have been proposed in this thesis.

In the second part, first we propose a model that gets raw input frames from videos as input and learn features in fewer layers having fewer parameters due to its pyramid structure. It is evaluated for action and dynamic scene recognition applications. It provides better results in case of Weizmann (AR) and MaryLand-in-the-wild (DSR) datasets. Whereas, shows comparable results with KTH (AR) and YUPENN (DSR) datasets. In our future focus, we are further verifying the generality of this model by evaluating it on recent larger and challenging datasets like UCF sports, Youtube action, and UT-Interaction datasets. This will help in proving benefits of using strictly pyramidal structure instead of non-pyramidal structure for learning a powerful model, since the model is aimed to obtain good performance despite the complexity and diversity of these datasets.

We extend *3DPyraNet* model to learn spatio-temporal features and classifying it with a linear-*SVM* classifier. Further, we show that the utilized fusion model is capable of learning powerful motion features by refining the sparse learned features, achieving competitive results with respect to current best methods on different video analysis benchmarks for action/scene recognition. A *DSR* system can provide relevant contextual information in many other applications of video analysis, e.g., scene context can improve human action recognition. Our future focus is on verifying the generality of our model by going deeper and testing it on the same as well as more recent larger and challenging datasets.

In the third part, we show the impact and power of pyramidal structure in a *CNN* model, by imposing it on state-of-the-art deep architecture i.e. *BVLC_Reference_Caffe*. This shows that, pyramid structure can enhance performance of state-of-the-art models as well as reduces not only number of parameters but also reduces solver memory size on disk for almost same or better results. In addition, a generalized criteria for designing a deep *CNN* architecture has been given which can be used as a starting point by many new researchers. Deep *CNN* has been deeply analyzed and we found that the last convolution layer has high impact. This approach can provide great difference in specific applications.

Performance on several datasets depict the impact and importance of pyramidal structure in *CNN* and *3DPyraNet*. We also want to stress that the results of all models evaluated in this thesis could potentially be improved by increasing the overall model size or a more thorough search for hyper kernels and neurons in fully connected layers. In a sense, this fact makes it even more surprising that the simple strict pyramidal model outperforms many existing approaches.

Another promising direction of our future research could be optimization of our *3DPyraNet*. It will assure the power by testing it with more real world datasets e.g. Sports 1Million videos dataset.

Acronyms

AR Action Recognition

BNN Biological Neural Network

BSC Body Surface Context

CNN Convolutional Neural Network

CRBM Convolutional Restricted Boltzman Machine

convGRBM Convolutional Gated Restricted Boltzman Machine

CV Computer Vision

CMC Cummulative Matching Characteristic

C3D Deep 3DCNN Features

CG Conjugate Gradient

CE Cross Entropy

DBN Deep Belief Network

DL Deep Learning

DSC Dynamic Scene Classification

DSR Dynamic Scene Recognition

DSU Dynamic Scene Understanding

FV Feature Vector

GD Gradient Descent

GMM Gaussian Mixture Model

GPU Graphical Processing Unit

GIST Gradient

GRBM Gated Restricted Boltzman Machine

GDMV Gradient Descent with Momentum and Variable Learning rate

GMC Grand Mother Cells

HMM Hidden Markov Model

HoF Histogram of Optical Flow

HoG Histogram of Oriented Gradient

IFV Improve feature vector

I-PyraNet Inhibatory Pyramidal Neural network

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IP Image Pyramids

LBP Local Binary Patterns

LDA Linear Discriminant Analysis

LGN Lateral Geniculate Nucleaus

LFW Labeled Faces in the Wild

LIPNet Lateral Inhibation Neural Network

LLC Local Linear Coding

LM Leveberg Marquadrt

LSTM Long Short Term Memory Network

LTRC Long Term Recurrent Convolutional Network

LNCP Locally Connected Neural Pyramid

ML Machine Learning

MSE Mean Square Error

mAP Mean Average Precision

MBH Motion Boundary Histogram

NN Neural Network

NIN Network-in-Network

OR Object Recognition

PCA Principal Component Analysis

PRe-ID Person Re-Identification

PNN Pyramidal Neural Networks

VPNN Variable Pyramidal Neural Network

VPNN-EA Variable Pyramidal Neural Network with Evolutionary Algorithm

PVC Primary Visual Cortex

PyraNet S. L Phung and A. Bouzerdoum Pyramidal Neural network

RBM Restricted Boltzmann Machine

RNN Recurrent Neural Network

RPROP Resilient Back Propagation

SA Statistical Aggregation

SOE Spatiotemporal Oriented Energy Features

SGD Stochastic Gradient Descent

SIFT Scale-invariant Feature Transform

STIP Space-time interest points

STRF Spatio-Temoral Regularity based Features

STPSC Spatio-Temoral Pyramid Sparse Coding

SPM Spatial Pyramid Matching

SPP Spatial Pyramid Pooling

SURF Speeded Up Robust Features

SVM Support Vector Machine

SWLD Spatial Weber's Local Descriptor

SCRF Segmentation and Classification with Receptive Field

ToSP Trajectory of Surface Patch

VLAD Vector of Locally Aggregated Descriptors

VC vector quantization

WS Weighted Sum

3DPyraNet 3D Pyramidal Neural network

3DCNN 3D Convolutional Neural network

Bibliography

[1] Zeiler, M., Fergus, R.: Visualizing and understanding convolutional networks. Computer Visionâ“ECCV 2014 (nov 2014) xiv, 55, 62, 63, 64, 68

[2] Derpanis, K.G., Lecce, M., Daniilidis, K., Wildes, R.P.: Dynamic scene understanding: The role of orientation features in space and time in scene classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012) 1306–1313 2, 3, 19, 20, 106, 107, 120

[3] Theriault, C., Thome, N., Cord, M.: Dynamic scene classification: Learning motion descriptors with slow features analysis. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2013) 2603–2610 2, 3, 106, 107

[4] Feichtenhofer, C., Pinz, A., Wildes, R.P.: Spacetime forests with complementary features for dynamic scene recognition. In: BMVC. (2013) 2, 106

[5] Shroff, N., Turaga, P., Chellappa, R.: Moving vistas: Exploiting motion for describing scenes. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2010) 1911–1918 2, 20

[6] Laptev, I., Pérez, P.: Retrieving actions in movies. In: Proceedings of the IEEE International Conference on Computer Vision. (2007) 3

[7] Schindler, K., Van Gool, L.: Action Snippets: How many frames does human action recognition require? 26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR (2008) 3

[8] Yang, X., Tian, Y.: Action Recognition Using Super Sparse Coding Vector with Spatio-temporal Awareness. In: Computer Vision – ECCV 2014. Volume 8690. (2014) 727–741 3

[9] Liu, W., Wang, Z., Tao, D., Yu, J.: Hessian Regularized Sparse Coding for Human Action Recognition. In: 21st International Conference, MMM 2015, Sydney, NSW, Australia, January 5-7, 2015, Proceedings, Part II. (2015) 502–511 3

[10] Melfi, R., Kondra, S., Petrosino, A.: Human activity modeling by spatio temporal textural appearance. Pattern Recognition Letters **34**(15) (November 2013) 1990–1994 3, 57, 78

[11] Cantoni, V., Petrosino, a.: Neural recognition in a pyramidal structure. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council **13**(2) (January 2002) 472–80 3, 6, 24, 26, 28, 29, 31, 41, 42, 74, 108

[12] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems. (2012) 1097–1105 3, 4, 17, 18, 43, 44, 46, 47, 50, 51, 55, 60, 62, 108, 122, 127, 128, 129, 130, 148

[13] Ji, S., Yang, M., Yu, K.: 3D convolutional neural networks for human action recognition. IEEE transactions on pattern analysis and machine intelligence **35**(1) (2013) 221–31 3, 4, 16, 54, 59, 74, 78, 100, 101, 102, 107, 119, 124

[14] Phung, S.L., Bouzerdoum, A.: A pyramidal neural network for visual pattern recognition. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council **18**(2) (March 2007) 329–43 3, 7, 31, 41, 42, 43, 108, 128, 130, 131

[15] a.a. Efros, a.C. Berg, Mori, G., Malik, J.: Recognizing action at a distance. Proceedings Ninth IEEE International Conference on Computer Vision (2003) 3

[16] Schüldt, C., Laptev, I., Caputo, B.: Recognizing human actions: A local SVM approach. Proceedings - International Conference on Pattern Recognition **3** (2004) 32–36 3, 97, 100, 102, 118, 119, 124

[17] Ballan, L., Bertini, M., Del Bimbo, A., Seidenari, L., Serra, G.: Effective codebooks for human action representation and classification in unconstrained videos. IEEE Transactions on Multimedia **14**(4 PART 2) (2012) 1234–1245 3

[18] Feichtenhofer, C.: Spacetime Forests with Complementary Features for Dynamic Scene Recognition. Bmvc2013 (2013) 2012 3, 21, 107, 120, 124

[19] Feichtenhofer, C., Pinz, A., Wildes, R.: Bags of spacetime energies for dynamic scene recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2014) 2681–2688 3, 107, 120, 121, 124

[20] Taylor, G.W., Fergus, R., LeCun, Y., Bregler, C.: Convolutional learning of spatio-temporal features. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6316 LNCS**(PART 6) (2010) 140–153 3, 15

[21] Freitas, N.D.: Deep Learning of Invariant Spatioâˆ’ Temporal Features from Video. In: Workshop on Deep Learning and Unsupervised Feature Learning in NIPS. (2010) 1–9 3, 16

[22] Le, Q.V., Zou, W.Y., Yeung, S.Y., Ng, A.Y.: Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2011) 3361–3368 3, 16

[23] D. Tran, L. Bourdev, R.F.L.T., Paluri, M.: Learning Spatiotemporal Features with 3D Convolutional Networks. In: ICCV, IEEE (jun 2015) 1725–1732 3, 18, 22, 59, 102, 108, 120, 122, 124

[24] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-Scale Video Classification with Convolutional Neural Networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition (June 2014) 1725–1732 3, 15, 17, 59, 108, 118, 121

[25] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324 3, 7, 41, 42, 43, 45

[26] Sermanet, P., LeCun, Y.: Traffic sign recognition with multiscale Convolutional Networks. The 2011 International Joint Conference on Neural Networks (2011) 2809–2813 3

[27] CireÅŸan, D., Meier, U., Masci, J., Schmidhuber, J.: Multicolumn deep neural network for traffic sign classification. Neural Networks **32** (aug 2012) 333–338 3

[28] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR **abs/1502.01852** (2015) 4

[29] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3) (2015) 211–252 4, 46, 47, 133

[30] Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. CoRR **abs/1506.02626** (2015) 5, 6, 108, 122, 127, 138, 140, 143, 144

[31] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. (2015) 1–9 6, 122, 127, 129

[32] Essen, D.C.V., Gallant, J.L.: Neural mechanisms of form and motion processing in the primate visual system. Neuron **13**(1) (1994) 1 – 10 6

[33] Fukushima, K.: Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks **1**(2) (January 1988) 119–130 6, 7, 24, 25, 26, 108

[34] Yeh, C.I., Xing, D., Shapley, R.M.: "Black" Responses Dominate Macaque Primary Visual Cortex V1. Journal of Neuroscience **29**(38) (sep 2009) 11753–11760 6

[35] Hubel, D.H., Wiesel, T.N.: Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology **195**(1) (1968) 215–43 6

[36] Lazebnik, S., Schmid, C., Ponce, J.: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06) **2** (2006) 2169–2178 7, 106, 128, 130

[37] Fan, H., Cao, Z., Jiang, Y., Yin, Q., Doudou, C.: Learning deep face representation. CoRR **abs/1403.2802** (2014) 7, 128

[38] Wang, P., Cao, Y., Shen, C., Liu, L., Shen, H.T.: Temporal pyramid pooling based convolutional neural networks for action recognition. CoRR **abs/1503.01224** (2015) 7, 128

[39] Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., eds.: Advances in Neural Information Processing Systems 27. Curran Associates, Inc. (2014) 568–576 7, 128

[40] Laptev, I., Lindeberg, T.: Space-time interest points. IEEE International Conference on Computer Vision (ICCV 03) (2003) 32 – 439 12

[41] Soomro, K., Zamir, A.R., Shah, M.: UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR **abs/1212.0402** (2012) 13

[42] Wang, H., Ullah, M.M., Klaser, A., Laptev, I., Schmid, C.: Evaluation of local spatio-temporal features for action recognition. In: Proc. BMVC. (2009) 124.1–124.11 doi:10.5244/C.23.124. 13

[43] Schindler, K., Gool, L.J.V.: Action snippets: How many frames does human action recognition require? In: CVPR, IEEE Computer Society (2008) 13

[44] Muhammad Muneeb Ullah, Sobhan Naderi Parizi, I.L.: Improving bag-of-features action recognition with non-local cues. International Conference on Computer Vision (ICCV 03) (2010) 1–8 13

[45] Song, Y., Zheng, Y.T., Tang, S., Zhou, X., Zhang, Y., Lin, S., Chua, T.S.: Localized multiple kernel learning for realistic

human action recognition in videos. Circuits and Systems for Video Technology, IEEE Transactions on **21**(9) (Sept 2011) 1193–1202 13

[46] Song, Y., Liu, S., Tang, J.: Describing trajectory of surface patch for human action recognition on rgb and depth videos. Signal Processing Letters, IEEE **22**(4) (April 2015) 426–429 13

[47] Song, Y., Tang, J., Liu, F., Yan, S.: Body surface context: A new robust feature for action recognition from depth videos. IEEE Transactions on Circuits and Systems for Video Technology **24**(6) (June 2014) 952–964 14

[48] Yu, G., Liu, Z., Yuan, J.: Discriminative orderlet mining for real-time recognition of human-object interaction. In: Computer Vision - ACCV 2014 - 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part V. (2014) 50–65 14

[49] Zhang, X., Zhang, H., Cao, X.: Action recognition based on spatial-temporal pyramid sparse coding. In: Pattern Recognition (ICPR), 2012 21st International Conference on. (Nov 2012) 1455–1458 14

[50] Muhammad Muneeb Ullah, Sobhan Naderi Parizi, I.L.: The representation and recognition of human movement using temporal templates. IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'97) (1997) 928â“934 14

[51] Blank, M., Gorelick, L., Shechtman, E., Irani, M., Basri, R.: Actions as space-time shapes. In: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. (2005) 1395–1402 Vol. 2 14

[52] Wang, L., Leckie, C.: Encoding actions via the quantized vocabulary of averaged silhouettes. IEEE International Confer-

ence on Computer Vision and Pattern Recognition (CVPR-10) (2010) 3657–3660 14

[53] Shao, L., Chen, X.: Histogram of body poses and spectral regression discriminant analysis for human action categorization. in Proceedings of the British Machine Vision Conference (BMVC) (2010) 3657–3660 14

[54] H. Qu, L.W., Leckie, C.: Action recognition using space-time shape difference images. in Pattern Recognition (ICPR), 2010 20th International Conference on. IEEE (2010) 3661 14

[55] X. Sun, M.C., Hauptmann, A.: Action recognition via local descriptors and holistic features. in IEEE Computer Society Conference in Computer Vision and Pattern Recognition Workshops (2009) 58–65 14

[56] Bengio, Y., Lecun, Y. In: Scaling learning algorithms towards AI. MIT Press (2007) 15

[57] Chen, B., Marlin, B.M., Ting, J.a.: Deep Learning of Invariant Spatio-Temporal Features from Video. NIPS Workshop (August) (2010) 1–9 15, 102, 124

[58] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia. MM '14, New York, NY, USA, ACM (2014) 675–678 15, 50, 51, 127, 133, 135, 136, 139, 144

[59] Hyvärinen, A., Hoyer, P.: Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces. Neural Comput. **12**(7) (July 2000) 1705–1720 16

[60] Baccouche, M., Mamalet, F., Wolf, C.: Sequential deep learning for human action recognition. Proc. Int. Conf. Human Be-

havior Understanding (HBU) (2011) 29–39 16, 18, 60, 100, 101, 102, 124

[61] Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. In: In ICML. (2010) 16

[62] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I. (2014) 818–833 17, 127

[63] Simonyan, K., Zisserman, A.: Two-Stream Convolutional Networks for Action Recognition in Videos. (June 2014) 1–11 18, 59, 60

[64] Donahue, J., Hendricks, L.A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T., Austin, U.T., Lowell, U., Berkeley, U.C.: Long-term Recurrent Convolutional Networks for Visual Recognition and Description. Cvpr (2015) 1–14 18, 60

[65] Ng, J., Hausknecht, M.: Beyond Short Snippets: Deep Networks for Video Classification. arXiv preprint arXiv: ... (2015) 4694–4702 18, 60

[66] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going Deeper with Convolutions. (sep 2014) 1–12 18, 51, 55, 60, 62, 69

[67] Feichtenhofer, C., Pinz, A., Wildes, R.: Dynamic Scene Recognition with Complementary Spatiotemporal Features. Pattern Analysis and Machine Intelligence, IEEE Transactions on **PP**(99) (2016) 1 19, 21, 59, 107, 120, 121, 124, 125

[68] Marszalek Marcin. Laptev Ivan, S.C.: Actions in Context. (i) (2009) 2929–2936 19, 20

[69] Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: Sun database: Large-scale scene recognition from abbey to zoo. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. (June 2010) 3485–3492 19, 106

[70] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., eds.: Advances in Neural Information Processing Systems 27. Curran Associates, Inc. (2014) 487–495 19, 106

[71] Quattoni, A., Torralba, A.: Recognizing indoor scenes (2009) 19

[72] Vaughan, C.L., Davis, B.L., O'Connor, J.C.: Dynamics of human gait. Human Kinetics Publishers, Leeds (England) (1992) 20

[73] Feichtenhofer, C., Pinz, A., Wildes, R.P.: Bags of spacetime energies for dynamic scene recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2014) 2681–2688 21

[74] Gangopadhyay, A., Tripathi, S.M., Jindal, I., Raman, S.: SA-CNN: Dynamic Scene Classification using Convolutional Neural Networks. (2015) 1–25 22

[75] Turk, M., Pentland, A.: Face recognition using eigenfaces. In: Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on. (Jun 1991) 586–591 23, 131

[76] D. Blei, A.N., Jordan, M.: Latent dirichlet allocation. in Journal of Machine Learning Research **3** (2003) 993–1022 23

[77] Hyvarinen, A., Hoyer, P.O.: Topographic independent component analysis as a model of v1 organization and receptive fields. in Journal of Neurocomputing **38**(40) (2001) 1307–1315 23

[78] Hoyer, P.O.: Topographic independent component analysis as a model of V1 organization and receptive " elds. **40** (2001) 1307–1315 24, 28, 29

[79] Bischof, H., Kropatsch, W.G.: Neural Networks vs Image Pyramids. Artificial Neural Nets and Genetic Algorithms **43**(222) (1993) 24

[80] Fukushima, K.: Neocognitron for handwritten digit recognition. Neurocomputing **51** (April 2003) 161–180 26

[81] Fukushima, K.: Artificial vision by multi-layered neural networks: neocognitron and its advances. Neural networks : the official journal of the International Neural Network Society **37** (January 2013) 103–19 27, 42, 43

[82] Mcquoid, M.R.: Neural ensembles: Simultaneous recognition of multiple 2-D visual objects. Neural Networks **6**(7) (jan 1993) 907–917 28

[83] Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET. (2015) 1–14 31

[84] cuda convnet: Locally-connected layer with unshared weights (May 2015) 32

[85] Fernandes, B.J.T., Cavalcanti, G.D.C., Ren, T.I.: A receptive field based approach for face detection. In: International Joint Conference on Neural Networks, IEEE (jun 2009) 803–810 39, 42, 43

[86] Rizzolatti, G., Camarda, R.: Inhibition of visual responses of single units in the cat visual area of the lateral suprasylvian

gyrus (clare-bishop area) by the introduction of a second visual stimulus. Brain Research **88**(2) (1975) 357–361 39

[87] Fernandes, B.J.T., Cavalcanti, G.D.C., Ren, T.I.: Nonclassical receptive field inhibition applied to image segmentation. Neural Network World **19**(1) (2009) 21–36 40, 42, 43, 128, 131

[88] Fernandes, B.J.T., Cavalcanti, G.D.C., Ren, T.I.: Lateral inhibition pyramidal neural network for image classification. IEEE transactions on cybernetics **43**(6) (dec 2013) 2082–91 40, 42, 43, 128, 131

[89] Soares, A.M., Fernandes, B.J.T., Bastos-Filho, C.J.A.: Pyramidal neural networks with evolved variable receptive fields. Neural Computing and Applications (2016) 40

[90] Uetz, R., Behnke, S.: Locally-connected hierarchical neural networks for gpu-accelerated object recognition. NIPS 2009 Workshop on Large-Scale Machine Learning: Parallelism and Massive Datasets, Whistler, Canada, December (December) (2009) 10–13 41, 42, 43

[91] Bengio, Y., Courville, A., Vincent, P.: Representation Learning: A Review and New Perspectives. (1993) (June 2012) 1–30 44

[92] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. (nov 2013) 49, 65

[93] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. arXiv preprint arXiv (2013) 1312.6229 49

[94] Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and Transferring Mid-level Image Representations Using Convolu-

tional Neural Networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition (June 2014) 1717–1724 49

[95] Oquab, M., Laptev, I., Learning, J.S., Mid-level, T., Bottou, L.: Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks To cite this version : Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In: IEEE conferenc eon computer Vision and Pattern Recognition. (2014) 49

[96] Lin, M., Chen, Q., Yan, S.: Network in network. CoRR **abs/1312.4400** (2013) 50, 51, 55, 122, 127, 129, 142, 144

[97] Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-Supervised Nets. (sep 2014) 1–10 50, 55

[98] : Internet Meme: we-need-to-go-deeper 52

[99] Le, Q.V., Ngiam, J., Chen, Z., Chia, D., Koh, P.W., Ng, A.Y.: Tiled convolutional neural networks. Advances in Neural Information Processing Systems 23 (2010) 1279–1287 55

[100] Wu, R., Yan, S., Shan, Y., Dang, Q., Sun, G.: Deep Image: Scaling up Image Recognition. Arxiv (2015) 12 55

[101] Maddalena, L., Petrosino, A.: The 3dSOBS+ algorithm for moving object detection. Computer Vision and Image Understanding **122** (May 2014) 65–73 57

[102] DiCarlo, J.J., Zoccolan, D., Rust, N.C.: How does the brain solve visual object recognition? Neuron **73**(3) (February 2012) 415–34 62

[103] Ba, J., Caruana, R.: Do deep nets really need to be deep? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., eds.: Advances in Neural Information Processing Systems 27. Curran Associates, Inc. (2014) 2654–2662 62

[104] Agrawal, P., Girshick, R., Malik, J.: Analyzing the Performance of Multilayer Neural Networks for Object Recognition. (July 2014) 62, 65, 66

[105] Gorelick, L., Blank, M., Shechtman, E., Irani, M., Basri, R.: Actions as space-time shapes. Transactions on Pattern Analysis and Machine Intelligence **29**(12) (December 2007) 2247–2253 97

[106] Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. ICML. Volume 30. (2013) 98

[107] Chaaraoui, A.A., Climent-Perez, P., Florez-Revuelta, F.: Silhouette-based human action recognition using sequences of key poses. Pattern Recognition Letters **34**(15) (2013) 1799 – 1807 Smart Approaches for Human Action Recognition. 99, 100

[108] Dollár, P., Rabaud, V., Cottrell, G., Belongie, S.: Behavior recognition via sparse spatio-temporal features. Proceedings - 2nd Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, VS-PETS **2005** (2005) 65–72 100, 102, 118, 119, 124

[109] Mukerjee, A.P.D.T.K.: Human Action Classification using 3D-Convolutional Neural Network. Technical report, Indian Institute of Technology Kanpur, Kanpur (2012) 101

[110] Klaser, A., Marszalek, M., Schmid, C.: A Spatio-Temporal Descriptor Based on 3D-Gradients. Proceedings of the British Machine Conference (2008) 99.1–99.10 102, 124

[111] Wang, H., Ullah, M.M., Klaser, A., Laptev, I., Schmid, C.: Evaluation of local spatio-temporal features for action recognition. BMVC 2009 - British Machine Vision Conference (2009) 124.1–124.11 102, 124

[112] Maninis, K., Koutras, P., Maragos, P.: Advances on action recognition in videos using an interest point detector based on multiband spatio-temporal energies. In: 2014 IEEE International Conference on Image Processing (ICIP), IEEE (oct 2014) 1490–1494 102, 124

[113] Scovanner, P., Ali, S., Shah, M.: A 3-dimensional sift descriptor and its application to action recognition. Proceedings of the ACM International Conference on Multimedia (MM 2007) (c) (2007) 357 102, 124

[114] Wang, H., Klaser, A., Schmid, C., Liu, C.L.: Action recognition by dense trajectories. In: CVPR 2011, IEEE (jun 2011) 3169–3176 102, 119, 124

[115] Weinland, D., Özuysal, M., Fua, P.: Making action recognition robust to occlusions and viewpoint changes. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6313 LNCS**(PART 3) (2010) 635–648 102, 124

[116] Perona, P.: A Bayesian Hierarchical Model for Learning Natural Scene Categories. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) **2** 524–531 106

[117] Vogel, J., Schiele, B.: Semantic modeling of natural scenes for content-based image retrieval. International Journal of Computer Vision **72**(2) (2007) 133–157 106, 107

[118] Szummer, M., Picard, R.: Temporal texture modeling. In: Proceedings of 3rd IEEE International Conference on Image Processing. Volume 3., IEEE 823–826 107

[119] Doretto, G., Chiuso, A., Wu, Y.N., Soatto, S.: Dynamic textures. International Journal of Computer Vision **51**(2) (2003) 91–109 107

[120] Ullah, I., Hussain, M., Muhammad, G., Aboalsamh, H., Bebis, G., Mirza, A.M.: Gender Recognition from Face Image with Local WLD Descriptor. 19th International Conference on Systems, Signals and Image Processing (IWSSIP) (April) (2012) 11–13 107

[121] Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L., Wang, Z.: Deep Fried Convnets. Iccv (2015) 1–10 108, 122

[122] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics. (2010) 112

[123] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In: Neural Networks: Tricks of the Trade. Springer (2012) 437–478 112

[124] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning Spatiotemporal Features with 3D Convolutional Networks. Proceedings in International Conference on Computer Vision (2015) 118, 120, 121, 125

[125] Theriault, C., Thome, N., Cord, M.: Dynamic scene classification: Learning motion descriptors with slow features analysis. In: Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. (June 2013) 2603–2610 120, 124

[126] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014) 127, 129

[127] Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In Dasgupta, S.,

Mcallester, D., eds.: Proceedings of the 30th International Conference on Machine Learning (ICML-13). Volume 28., JMLR Workshop and Conference Proceedings (May 2013) 1058–1066 127

[128] Mairal, J., Koniusz, P., Harchaoui, Z., Schmid, C.: Convolutional kernel networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., eds.: Advances in Neural Information Processing Systems 27. Curran Associates, Inc. (2014) 2627–2635 127

[129] He, X., Luo, S., Tao, D., Xu, C., Yang, J., Hasan, M.A., eds.: MultiMedia Modeling. Volume 8936 of Lecture Notes in Computer Science. Springer International Publishing, Cham (2015) 127, 130

[130] Kaiming, H., Xiangyu, Z., Shaoqing, R., Jian, S.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: The IEEE International Conference on Computer Vision (ICCV). (December 2015) 127

[131] Bischof, H., Kropatsch, W.: Neural networks versus image pyramids. In Albrecht, R., Reeves, C., Steele, N., eds.: Artificial Neural Nets and Genetic Algorithms. Springer Vienna (1993) 145–153 127

[132] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. (1998) 2278–2324 128, 129

[133] Zeiler, M.D., Fergus, R.: Stochastic pooling for regularization of deep convolutional neural networks. CoRR **abs/1301.3557** (2013) 128, 144

[134] Collins, M.D., Kohli, P.: Memory bounded deep convolutional networks. CoRR **abs/1412.1442** (2014) 129, 141, 143, 144

[135] Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. (May 2013) 6655–6659 129

[136] Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting Parameters in Deep Learning. (2013) 1–9 129

[137] Fukushima, K.: Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks **1**(2) (1988) 119 – 130 130

[138] Hubel, D.H.: The visual cortex of the brain. Sci. Amer. **209** (1963) 54â“62 130

[139] Masci, J., Meier, U., Fricout, G., Schmidhuber, J.: Object Recognition with Multi-Scale Pyramidal Pooling Networks. arxiv (jul 2012) 130

[140] LeCun, Y., Cortes, C.: MNIST handwritten digit database. (2010) 132

[141] Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report (2009) 132, 133

[142] Gray, D., Brennan, S., Tao, H.: Evaluating appearance models for recognition, reacquisition, and tracking. In: 10th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS). (09/2007 2007) 134

[143] Vezzani, R., Baltieri, D., Cucchiara, R.: People reidentification in surveillance and forensics: A survey. ACM Comput. Surv. **46**(2) (December 2013) 29:1–29:37 134

[144] Yi, D., Lei, Z., Liao, S., Li, S.Z.: Deep metric learning for person re-identification. In: ICPR. (2014) 134, 147

[145] Lee, C., Xie, S., Gallagher, P.W., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015. (2015) 142, 144

[146] Gray, D., Brennan, S., Tao, H.: Evaluating appearance models for recognition, reacquisition, and tracking. In: In IEEE International Workshop on Performance Evaluation for Tracking and Surveillance, Rio de Janeiro. (2007) 146

[147] Xu, Y., Lin, L., Zheng, W.S., Liu, X.: Human re-identification by matching compositional template with cluster sampling. In: Proceedings of the IEEE International Conference on Computer Vision. (2013) 3152–3159 146

[148] Zhao, R., Ouyang, W., Wang, X.: Learning mid-level filters for person re-identification. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2014) 144–151 147

[149] Wang, Y., Lu, W.F., Fuh, J.Y.: Sampling Strategies for 3D Partial Shape Matching and Retrieval Using Bag-of-Words Model. Computer-Aided Design and Applications **11**(1) (January 2014) 43–48 149