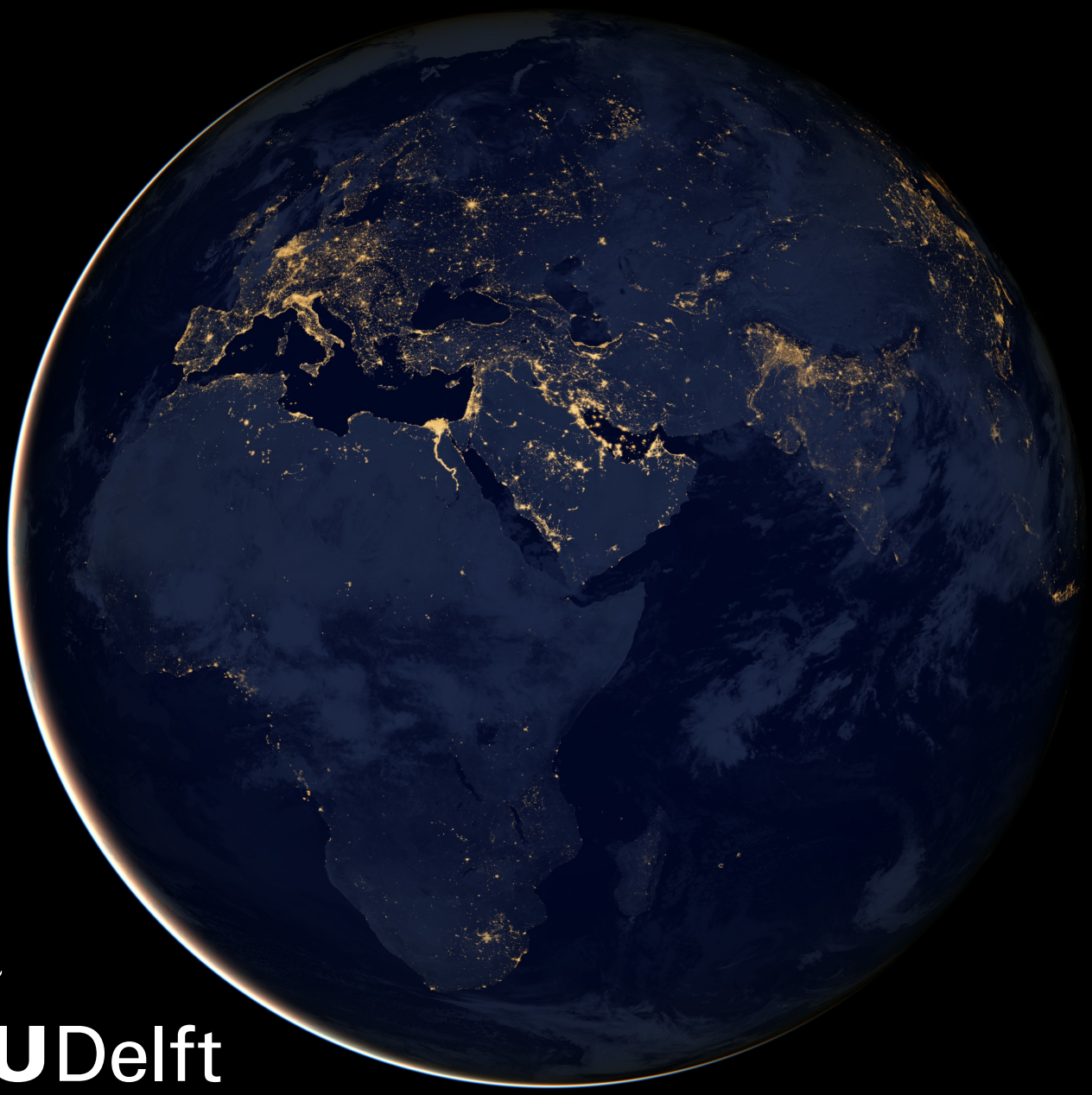
Group-Equivariant Video Action Recognition

Making action-recognition networks equivariant to temporal and spatial transformations

Debadeep Basu



Delft University of Technology

TU Delft

Group-Equivariant Video Action Recognition

Making action-recognition networks equivariant to temporal and spatial transformations

by

Debadeep Basu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday, December 22, 2021 at 15:15.

Student number:	5089107	
Project duration:	December, 2020 – December 2021	
Thesis committee:	Prof. dr. ir. J. C. van Gemert,	TU Delft, Supervisor and Chair
	Dr. C. Lofi,	TU Delft, External Core Member
	O. Strafforello,	TU Delft, PhD Student

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report documents the research on applying group equivariance to video action recognition tasks for my Master Thesis to obtain the Master of Science degree from Delft University of Technology. The main content of the report is the scientific paper as the first chapter. It is followed by chapters that explain supplementary details about the work. My work was done with the Pattern Recognition and Bioinformatics research group, under the guidance of Dr. (Prof). Jan van Gemert, head of the Computer Vision Lab.

Firstly, I would like to express my sincere gratitude to Jan van Gemert for his guidance and feedback throughout the duration of the project. Secondly, I would like to thank my daily supervisor Ombretta, who had been encouraging me, guiding, and helping me through the obstacles along the way.

I would also like to take this opportunity to thank my parents, my friends for their constant support and guidance through the both good days and the bad ones. A special mention to my roommate and best friend Ramya, for believing in me constantly and being a beacon of hope, through what I can only describe as a long and difficult road.

Finally, I would like to express my appreciation to Dr. Cristoph Lofi for showing interest in my work, and accepting to be a part of my evaluation committee.

Debadeep Basu
Delft, December 2021

Contents

Preface i

1 Scientific Article 1

2 Basics of Deep Learning 13
2.1 Feed-forward Networks . . . 13
2.1.1 Activation function . . . 14
2.2 Optimization . . . 14
2.2.1 Stochastic Gradient Descent . . . 15
2.2.2 Learning rate decay . . . 16
2.3 Convolutional Neural Networks . . . 16
2.3.1 Convolution operation . . . 17
2.3.2 Feature detection using kernels . . . 18
2.3.3 Pooling . . . 18

3 Action Recognition 21
3.1 Action recognition Datasets . . . 21
3.1.1 HMDB-51 . . . 21
3.1.2 UCF-101 . . . 21
3.1.3 Kinetics . . . 21
3.1.4 Moments in time . . . 22
3.1.5 Something-Something . . . 22
3.2 Common architectures for Action Recognition . . . 22
3.2.1 Video Representations using Long Short-Term Memory Networks . . . 22
3.2.2 Optical Flow . . . 22
3.2.3 Two-Stream CNNs . . . 22
3.2.4 3D Convolutional Networks . . . 25
3.2.5 3D Residual Networks . . . 27

4 Group Equivariant Convolutional Networks 28
4.1 Equivariance vs Invariance in CNNs . . . 28
4.2 Symmetry . . . 30
4.3 Groups . . . 30
4.3.1 2D Groups . . . 30
4.3.2 Functions on groups . . . 31
4.4 Group Equivariant Networks . . . 32
4.4.1 Translational Equivariance with Groups . . . 32
4.4.2 Group Equivariant Convolutions . . . 33
4.4.3 Deeper G-Convolutions . . . 33
4.4.4 Implementation . . . 33
4.5 Three Dimensional Group Convolutions . . . 34
4.5.1 3D Implementation . . . 35

Bibliography 37

List of Figures 40

List of Tables 42

A RNNs and LSTMs 43
A.1 A Brief Recap about Recurrent Neural Networks . . . 43
A.2 A Brief Recap about Long Short-Term Memory Networks . . . 43

B Task Division 46

1

Scientific Article

Group Equivariant Video Action Recognition

Debadeep Basu
Delft University of Technology
Delft, The Netherlands

Ombretta Strafforello (Supervisor)
Delft University of Technology
Delft, The Netherlands

Jan C. van Gemert (Supervisor)
Delft University of Technology
Delft, The Netherlands

Abstract

This work applies the theory of group equivariance to the domain of video action recognition replacing standard 3D convolutions with group convolutions which are equivariant to temporal direction, and multiples of 90 degree spatial rotations. We propose a temporal direction symmetry group T2, and extend the standard planar rotations group to three dimensions to form a 3D group which is equivariant to discrete 90 degree spatial rotations. We analyse the efficacy of using these 3D-G-CNNs as drop-in replacements in 3D networks by evaluating on synthesized datasets containing handwritten MNIST digits moving over a black background, as well as popular action recognition datasets UCF-101 and HMDB-51, and comparing the results against the performance of the standard 3D CNNs on the datasets.

1. Introduction

Convolutional Neural Networks(CNNs) are the de-facto standard for most computer vision domains and achieve state-of-the-art performance on tasks such as image classification, video action recognition, object detection, instance segmentation etc. One of the major contributors to the success of CNNs is the concept of translational equivariance, which ensures that if the input data is translated, the corresponding activations of the network in all layers translate in the same way. Cohen and Welling [1] generalise the equivariance in CNNs to larger symmetry groups consisting of planar rotations and reflections, allowing CNNs to be equivariant to rotations and mirror reflections in images as well. Group equivariant convolutions have since been extended to larger symmetry groups with 6-fold rotational equivariance [2], Steerable filters [3, 4] and spatial volumes [5, 6, 7], and have achieved improved performance over standard CNNs in various computer vision tasks such as image classification [1, 2, 3, 4], instance segmentation [8, 9], siamese tracking [10], and classification on volumetric medical data [5, 7].

In this paper, we apply the concept of Group equivariance to the field of video action recognition using 3D CNNs and analyse whether replacing standard 3D convolutions with 3D group equivariant convolutions consisting of temporal reverse symmetry groups and spatial rotation symmetry group makes action recognition networks equivariant to time-reversal and spatial rotations, assuming that for all transformations, the label corresponding to the video is preserved. We evaluate and compare the equivariant CNNs to the baseline 3D CNN architectures trained with and without data augmentation, using a synthesised dataset consisting of handwritten MNIST digits moving over a black background. We also evaluate the group equivariant CNNs on small but popular action recognition datasets UCF-101[11], and HMDB-51 [12] and compare them to the baseline 3D architectures. Figure 1 shows an example of a group convolution pipeline for video action recognition, taking a temporal direction equivariant group, and a shallow 2-layer CNN.

To summarise, the main contributions of the paper are as follows:

- We propose a temporal direction equivariant convolution layer using the concepts of group equivariance, which can be used in place of standard 3D convolutions in video action recognition tasks to make 3D networks equivariant to time-reversed videos, without the need for data augmentation.
- We extend the base 2D rotation equivariant group, introduced by Cohen and Welling [1], to 3D, and create a spatial rotation equivariant layer convolution layer to make 3D networks equivariant to spatial rotations

by multiples of 90 degrees in videos, without need for data augmentation.

- We show that temporal direction and spatial rotation equivariant 3D CNNs display improved performance over the standard baseline 3D-CNN models for video action recognition on UCF-101 and HMDB-51.

2. Related Work

2.1. Action recognition using 3D CNNs

3D CNNs perform convolution operations on not only the spatial dimension but the temporal dimension as well, thereby being able to learn representations of actions from videos [13, 14, 15]. While some networks utilize optical flow streams (besides the RGB stream) as an additional input to learn the temporal representations in a video [13, 16], recent publications have shown that 3D CNNs can be quite robust in learning spatio-temporal representations using only the RGB stream. Ji et al. [17] first introduced the 3D CNN for human action recognition tasks, aiming to capture temporal information from the stacked input frames, thereby laying the groundwork for using 3D CNNs for action recognition. Following on, Tran et al. [14] introduced C3D, a deeper variant of the 3D CNN introduced by Ji et al. [17], based on the VGG16 architecture. The C3D model showed strong generalization capabilities despite not being satisfactory on standard benchmarks. Carreira and Zisserman [13] cemented the 3D CNN as a state-of-the-art with the introduction of I3D, as well as a new large-scale dataset called Kinetics. I3D was able to adapt image classification architectures for use in 3D CNNs by inflating a 2D model pre-trained on ImageNet [18] to 3D. An I3D, pre-trained on Kinetics-400, achieved an accuracy of 95.6% on UCF-101 and 74.8% on HMDB51. Carreira and Zisserman [13], also integrated a temporal stream based on optical flow with the I3D network and showed significant improvement in performance with 98.0% on UCF-101 and 80.9% on HMDB51.

Using the Kinetics dataset introduced in [13], Hara et al. [15] replicated the successful history of deep 2D CNNs pre-trained on ImageNet, for video action classification. They extended the 2D ResNet architecture, replacing the 2D convolution kernels with their 3D counterparts, creating the 3D ResNet with various depts(18 to 200). They showed that like 2D ResNets trained on ImageNet, 3D Resnets pre-trained on Kinetics outperform complex 2D networks on both the UCF-101 and the HMDB-51 datasets. In our work, we use 3D CNNs as backbone architectures to replace standard 3D convolution layers with 3D group equivariant layers and evaluate them.

2.2. Temporal information in video classification

Temporal information plays an important role in the sphere of video classification, particularly in Arrow of Time(AoT) classification - determining whether a video is playing in forward or reverse, and has found significant utility in pre-training video understanding models. [19]. The importance of temporal information in action classification has also been displayed by recent convolutional networks, which utilize optical flow as an input along with the spatial frames(RGB) to better model the temporal features in videos, and tend to show increased performance [13, 16]. Although recent networks can achieve great performance on benchmark datasets such as the UCF-101 [11], HMDB-51 [12], or Kinetics [20], Sevilla-Lara et al. [21] observed that many action classes in these video datasets could be classified without explicitly requiring temporal information. They addressed the problem by creating a novel video recognition benchmark called *Temporal Dataset*, consisting of human-annotated classes where understanding temporal information was a requirement for successful classification. In this paper, we do not utilize an explicit temporal stream and use networks that learn action representations using only RGB frames as inputs, but we apply the idea of a dataset that requires good temporal understanding by creating a dataset whose class labels are not discernable from a single frame.

2.3. Homogeneous action label transformations

Changing the temporal order of an action in a video clip can often lead to a different action. However, as the RGB frames of the video clip remain the same for both actions, a network would be able to learn both action labels using the same video clip. Price and Damen [22] applied video transforms such as horizontal flipping, temporal reversal on their video clips to identify homogeneous label transforms - transformations that either modify or maintain the labels of videos in each class within the dataset. They evaluate the approach of discovering temporal order invariant classes [1]. Their approach, evaluated on the Jester dataset [23] and the Something-something dataset [24], and showed that class labels could be learnt from their equivariant pair classes. For instance, *zooming into something* can be learnt from *zooming out of something*. In our work, we consider only *Label preserving transforms* - transformations on videos that do not change the action or the label associated with it.

[1] Classes which, when transformed, either preserve their label, pairs of equivariant classes which exchange their labels, or labels novel to the dataset [22].

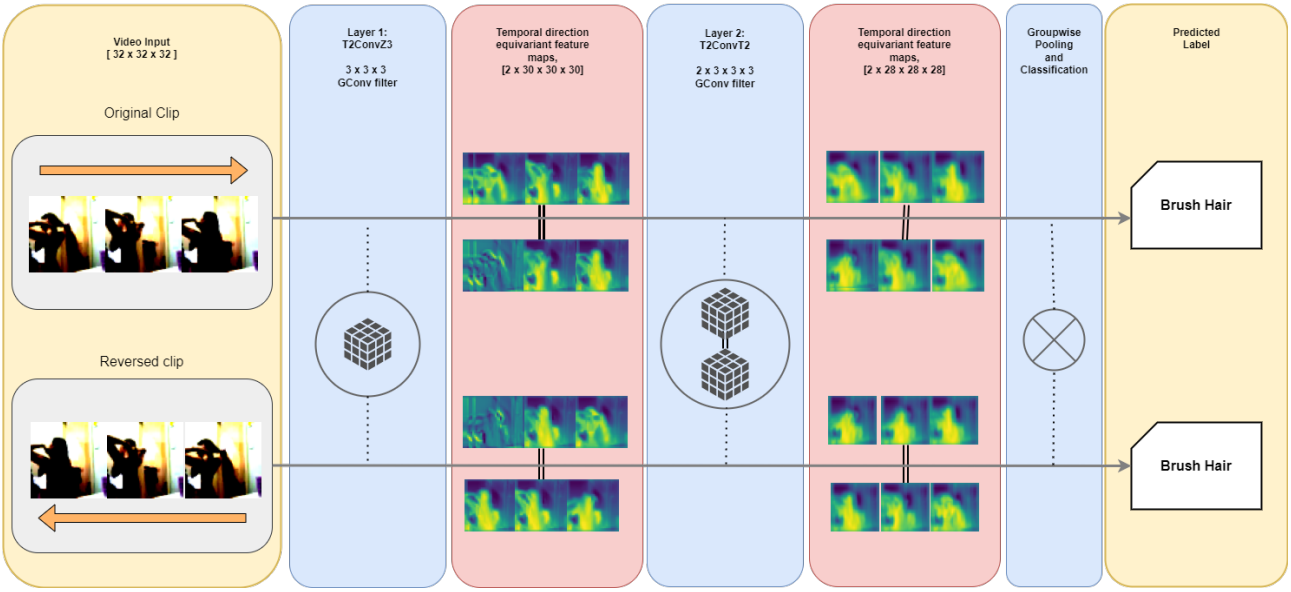


Figure 1: Overview of a group equivariant convolutional network pipeline shown using a shallow 2-layered G-CNN with a temporal direction equivariant group T2. The blue boxes represent group actions, the yellow ones represent input and output layers, and the red ones represent intermediate feature maps. Group convolutions occur in layers 1 and 2, with a group pooling in the penultimate layer. The top input shows 3 frames of the original clip taken from the HMDB-51 dataset with the label *Brush Hair*, and while the bottom input shows 3 frames of the reversed clip. The corresponding feature maps are shown after the group convolution operations and consist of two symmetry groups. As an output of the network, we see that the original clip and the reversed clip are classified correctly.

2.4. Group equivariant CNNs

2.4.1 Equivariance properties of CNNs

Equivariance property is a property of functions where the result of two functions f and g operating on an input x produce the same result when f is operated first followed by g or vice versa. Therefore, for an input x, the functions f(g(x)) and g(f(x)) yield the same result. In the context of neural networks and transformations, this would mean that performing a transformation and then a convolution operation on a certain input image is the same as first performing a convolution operation followed by a transformation of the resultant state.

2.4.2 Group equivariance

Although, by their very nature, CNNs provide translational equivariance, they are not equivariant or invariant to other transformations such as rotations and reflections. Group Equivariant Convolutional Networks [1] was proposed as a solution to the problem. Besides translational equivariance, G-CNNs show equivariance to rotations and reflections and a higher degree of weight sharing than standard CNNs. Weight sharing allows for learning a much larger set of possible symmetries without increasing the number of learnable parameters of the network, thereby making the network more data and resource-efficient over standard CNNs.

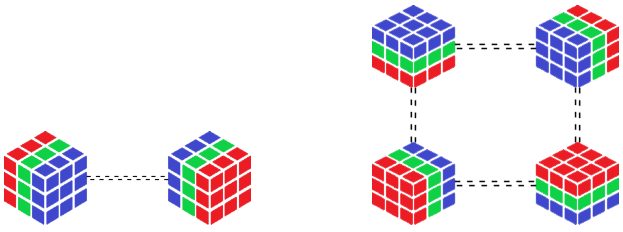
In a subsequent publication, the authors extended the G-CNNs for three-dimensional inputs by introducing 3D G-CNNs [5] consisting of octahedral and square prism symmetry groups. The networks were trained and tested with volumetric CT scan data and showed significant data and resource efficiency, producing a similar performance to networks trained with nearly ten times more data.

In our work, we use the concepts of group equivariance to create two symmetry groups and augment our backbone architecture with 3D group convolutions and attempt to make our network equivariant to translations and discrete spatial rotations, as well as equivariant to reflections in the temporal plane (such as time-reversed videos).

3. Methodology

3.1. T2 - The temporal direction equivariant group

The T2 group consists of the compositions of translations and mirror reflection along the spatial plane. The group can be parameterized in terms of four integers:



(a) T2 (b) P43D

Figure 2: Graphical representation of the defined 3D groups and their effects on canonical $3 \times 3 \times 3$ filters with 2 nodes for the T2 group, and 4 nodes for the P43D group. Each node corresponds to a symmetry transformation $g \, \epsilon \, G$.

z, u, v, w, and is represented by the homogeneous matrix:

$$g(z,u,v,w) = \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & -1^z & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where $z \, \epsilon \, \{0, 1\}$ and $(u, v, w) \, \epsilon Z3$, and the operation on the group is a matrix multiplication. The group TR operates on spatio-temporal coordinates in Z3 by taking the product of the group symmetry matrix g and the homogeneous 3D coordinate vector representing a spatio-temporal voxel. Figure 2a provides a representation of the T2 group, with the identity cubic filter(left) and the transformed filter(right). We notice that the cube is mirrored on the temporal axis under the transformation while preserving the spatial orientations. We visualize the effect of the filter group on a stack of frames in Figure 3, with the coordinates representing the pixel positions in each frame in the video, and see that the order of the frames is reversed along the temporal axis.

3.2. P43D - The spatial rotation equivariant group

The P43D group consists of the compositions of translations and 90-degree rotations on the spatial plane, about the centre. It is a 3D extension of the p4 feature map introduced by Cohen and Welling [1] and contains four elements, each representing a rotation by 90 degrees and the identity element. The group can be parameterized in terms of four integers: r, u, v, w, and is represented by the homogeneous matrix:

$$g(r,u,v,w) = \begin{bmatrix} cos(\frac{\pi}{2}) & -sin(\frac{\pi}{2}) & 0 & u \\ sin(\frac{\pi}{2}) & cos(\frac{\pi}{2}) & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where $0 \leq r < 4$ and $(u, v, w) \, \epsilon \, Z3$, and the operation on the group is a matrix multiplication. The group P43D operates on spatio-temporal voxel coordinates in Z3 by matrix multiplication. We represent the P43D group in Figure 2b

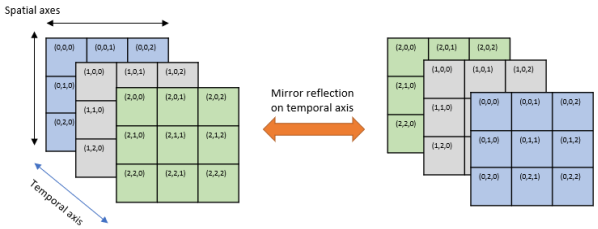


Figure 3: The temporal direction equivariant group visualized by transforming a stack of three frames. The stacked frames on the left represent the identity element, which upon being transformed by a mirror reflection on the temporal axis, forms the element on the right, representing the element of the group with the order of frames reversed. The colour of the frames can easily help visualize this. In the identity element, the green frame is nearest to the reader, and the blue frame is farthest, whereas, in the reversed element, the green frame is the farthest while the blue frame is the nearest. We can also see that the spatial orientations in the transformed element do not differ from the identity element.

as four pure 90 degree spatial rotations on a spatiotemporal cube about its center, while leaving its temporal dimension untransformed. This leads to four orientations of the cube representing the identity, and three rotated versions representing 90, 180 and 270 degree spatial rotations respectively. We see a stack of frames representation in Figure 4 where the spatial coordinates rotated, leaving the order of the frames unchanged.

3.3. 3D Group-Equivariant Convolutions

We write the Group convolution operation as defined in [1], as a composition of a translation $t \, \epsilon \, Z3$, and a transformation $s \, \epsilon \, G$, where G is the symmetry group. The transformation s leaves the origin invariant and is called the stabilizer. In the case of the T2 group, the transformation s is a mirror transform along the temporal axis. We can therefore write the convolution operation using an input spatiotemporal volume f, and the 3D filter ψ, as:

$$f \star \psi(ts) = \sum_{h \, \epsilon \, X} \sum_{k} f_k(h) L_t[L_s \psi_k(h)] \tag{3}$$

where L_t represents translations performed on the feature map, L_s represents the transformation on the feature map, k represents the k^{th} filter in the filter bank, $X = Z3$, or the voxel space of the feature map in the first layer of the network, and $X = G$ in the subsequent layers.

To compute the result of the convolution operation, the filter ψ is first transformed with L_s consisting of all transformations in the group G, followed by convolution on the input f.

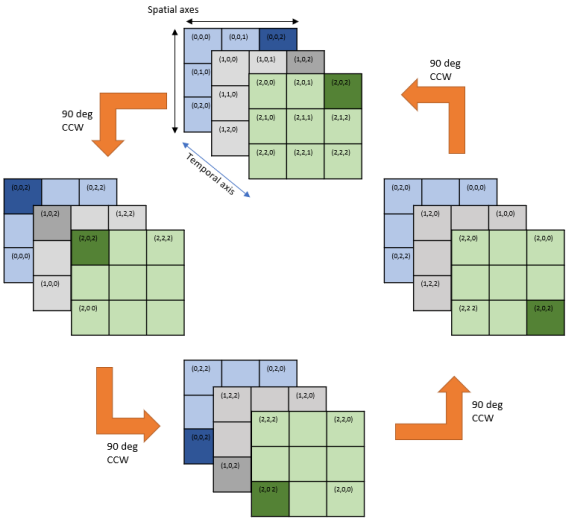


Figure 4: The spatial rotation equivariant group visualized by transforming a stack of three frames. The stack of frames at the top represents the identity element, which is transformed by four spatial rotations on the spatial domain forming the three other elements, each differing from the previous by a 90-degree counter-clockwise rotation. The rotations can be visualized by focusing on the darkened pixels in the frames. With each 90-degree rotation, the position of the pixel changes.

A detailed explanation of the group convolution process can be found in the supplementary material.

3.3.1 Transformed feature maps

The filters at any layer of a group equivariant 3D CNN are of the shape $K^l \times K^{l-1} \times S^{l-1} \times n \times n \times n$, where K_l is the number of channels in the current layer, n represents the translational extent of the filter, and S^{l-1} represents the number of transformations in the symmetry group G, which are 1, 2 and 4 for the Z3, T2 and P43D groups respectively. Performing a convolution operation with each filter produces a set of 2 feature maps for the T2 groups and 4 feature maps for the P43D feature maps.

4. Experiments

4.1. Network architectures

We compare the P43D and T2 groups and standard 3D Convolutions($Z3$) on the datasets using two baseline architectures. The first baseline is a simple 3D-CNN, consisting of a sequence of 3 non-strided $3 \times 3 \times 3$ convolutional layers, and 1 non-strided $1 \times 1 \times 1$ convolutional layers, with a 3D max pool layer after the first convolution, subsampling using an average pooling after the second and third layers, and an adaptive average pooling layer before the final fully-connected layer. All the convolution layers are followed by non-linear activations using Rectified Linear Unit(ReLU) modules. The second baseline is a 3D Residual network [15] with 18 layers. The network includes an initial 3D convolution layer followed by four stages consisting of 2 blocks each before the final fully connected layer. Each block consists of 2 convolutional layers with a kernel size of $3 \times 3 \times 3$, with ReLU activations after each convolution. Except for the first block, the remaining three blocks also consist of a downsampling layer with a single $1 \times 1 \times 1$ convolution layer using a stride of 2. The network also consists of a 3D max pool after the first convolution and a 3D Adaptive average pool before the final fully connected layer.

To construct the Group Equivariant 3D CNNs, we replace the standard 3D Convolution layers in both baseline architectures with the respective G-CNN modules and reduce the number of filters in each layer by a factor of $\sqrt{H}$, where H denotes the number of elements in the corresponding symmetry group. For the T2 group, $H = 2$, while for the P43D group, $H = 4$. We also add a global average pooling over the group orientation channels before the fully connected layer. We thereby obtain four 3D G-CNNs from our two baseline architectures - the 4-layer T2CNN, the 4-layer P43DCNN, the T2-3D ResNet-18 and the P43D-ResNet-18. The description of the layers of the networks based on the 4-layer CNN can be seen in Table 1. Table 2 shows the size of resultant 3D feature maps from each layer in the baseline Z3CNN, the T2CNN and the P43DCNN, along with the orientation channels for the 3D G-CNNs.

4.2. Datasets

4.2.1 Moving MNIST

The moving MNIST digits dataset is created by moving handwritten digits from the MNIST dataset [25], over a black background. The task associated with the dataset is to identify the digit in the video. The dataset is split into training, validation, and testing sets of 6000, 1000 and 3000 videos.

4.2.2 Moving MNIST-Dir

We also create another dataset using the handwritten digits of the MNIST dataset [25] moving over a black background with class labels representing the directional plane of movement of the digits within the square frame. The dataset consists of 10000 videos distributed evenly across four classes - *Horizontal, Vertical, Diagonal Left* and *Diagonal Right*. The dataset consists of 5000 videos for training, 2000 for validation and 3000 for testing.

Name	Z3CNN 687K parameters		T2CNN 676K parameters		P43DCNN 683K parameters	
	Layer	C_{in}, C_{out}	Layer	C_{in}, C_{out}, H	Layer	C_{in}, C_{out}, H
conv1	Conv3D	(3, 64)	T2ConvZ3	(3, 45, 2)	P43DConvZ3	(3, 32, 4)
maxPool	MaxPool3D	(64, 64)	MaxPool3D	(45, 45, 2)	MaxPool3D	(32, 32, 4)
conv2	Conv3D	(64, 128)	T2ConvT2	(45, 90, 2)	P43DConvP43D	(32, 64, 4)
conv3	Conv3D	(128, 128)	T2ConvT2	(90, 90, 2)	P43DConvP43D	(64, 64, 4)
smoothing	AveragePool3D	(128, 128)	AveragePool3D	(90, 90, 2)	AveragePool3D	(64, 64, 4)
conv3	Conv3D	(128, 128)	T2ConvT2	(90, 90, 2)	P43DConvP43D	(64, 64, 4)
adaptiveAvg	AdaptiveAvgPool3D	-	AdaptiveAvgPool3D	-	AdaptiveAvgPool3D	-
fc	Linear	(128, N)	Groupwise Avg Pool	-	Groupwise Avg Pool	-
			Linear	(90, N)	Linear	(64, N)

Table 1: 4-Layer CNN architecture descriptions showing the baseline $Z3$ CNN architecture, and the representation of the model with convolution layers replaced with the respective G-CNN layers. The resulting T2CNN and P43DCNN have 2 and 4 orientation channels, and have $K_l * \sqrt{H}$ feature maps, where K_l is the number of feature maps at a given layer, and H represents the number of elements in the symmetry group. N in the final layer(fc) represents the number of classes in the dataset the network is trained on.

Name	Filter size	Stride	Feature map size ($S_l, D_{out}, H_{out}, W_{out}$)		
			Z3CNN	T2CNN	P43DCNN
conv1	3 x 3 x 3	1	(32, 32, 32)	(2, 32, 32, 32)	(4, 32, 32, 32)
maxPool	3 x 3 x 3	2	(16, 16, 16)	(2, 16, 16, 16)	(4, 16, 16, 16)
conv2	3 x 3 x 3	1	(16, 16, 16)	(2, 16, 16, 16)	(4, 16, 16, 16)
conv3	3 x 3 x 3	1	(16, 16, 16)	(2, 16, 16, 16)	(4, 16, 16, 16)
smoothing	2 x 2 x 2	2	(8, 8, 8)	(2, 8, 8, 8)	(4, 8, 8, 8)
conv3	1 x 1 x 1	1	(8, 8, 8)	(2, 8, 8, 8)	(4, 8, 8, 8)
adaptiveAvg	-	-	(1, 1, 1)	(2, 1, 1, 1)	(4, 1, 1, 1)
groupPool	-	-	N/A	(1, 1, 1, 1)	(1, 1, 1, 1)

Table 2: Feature map output sizes for the layers in the three versions of the 4-layer CNN. S_l represents the number of symmetries generated, while $D_{out}, H_{out}, W_{out}$ represent the output depth, height and width respectively.

4.3. Evaluating with transformed datasets

To test whether the networks are indeed equivariant to group transformations, we evaluate the models over multiple test runs using the untransformed dataset and a transformed version of the dataset. The transformations which we use during the process of evaluation are as follows:

- **Temporal Reverse:** The order of the frames of the videos in the dataset are reversed. For example, if an object is translated from left to right in the original dataset, the object moves from left to right in the transformed dataset. In the subsequent sections, the test results on a reversed dataset are denoted by the term **treverse**
- **Spatial Rotations:** The frames of the videos in the dataset are rotated counter-clockwise by an angle specified during evaluation. As our symmetry consists of three 90 degree rotations, our transformations on the test are restricted to 90-degree rotations as well. The test results on the spatially rotated videos are represented by the terms **r90, r180**, and **r270**, where the number value represents the degree of counter-clockwise rotation applied on the dataset.

The base test set with no transformations applied to it during evaluation is represented by the term **original**. We visualise the effect of the transformations on videos using the Moving MNIST dataset in Figure 5.

4.4. Evaluating equivariance to temporal direction in videos in standard CNNs

We evaluate whether standard CNNs are equivariant to temporal direction and spatial rotations in videos. We train the 4-layer Z3CNN and a baseline 3D-ResNet-18 on the videos of the Moving MNIST-Dir dataset. Training was conducted for 50 epochs on the 4-layer Z3CNN with an initial learning rate of 0.1, with the learning rate being reduced by a factor of 0.1 after 20 and 40 epochs. The 3D-ResNet-18, with a comparatively greater number of learnable pa-

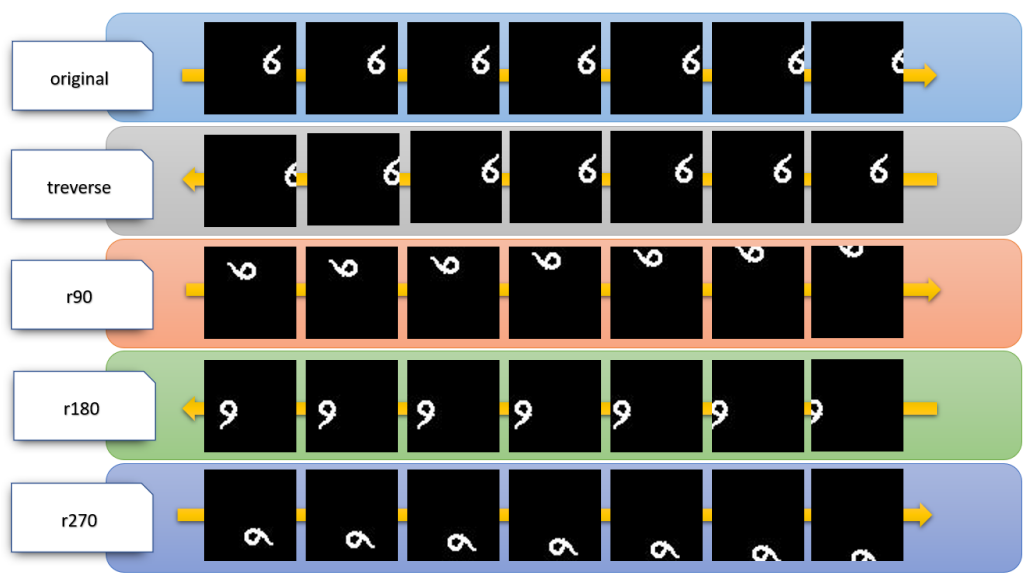


Figure 5: Visualisation of applying transformations on the video frames of the handwritten digit 6 moving from left to right over a black background. The term **original** represents the video with no transformations applied to it, **treverse** represents the video with reversed order of frames, **r90**, **r180** and **r270** represent a counter clockwise spatial rotation of 90 degrees, 180 degrees and 270 degrees respectively. The yellow arrow represents the temporal direction of the video relative to the order of video frames in **original**.

Model	**Type**	**Top-1 Accuracy(%)**	
		original	**treverse**
Z3CNN	4-layer CNN	100	24.6
Z3-ResNet-18	3D ResNet-18	100	24.13

Table 3: Best Top-1 accuracy obtained on the Moving MNIST-Dir dataset for the standard 4-layer CNN and the standard 3D-ResNet-18. The evaluation is done on the test sets separately with reverse transformation(treverse) and without transformation(original)

rameters, was trained for 100 epochs with an intial learning rate of 0.1, reduced by a factor of 0.1 after 25, 50, and 75 epochs. We evaluate the best Top-1 accuracy on the original test dataset and the reversed test dataset. Table 3 summarises the results for the evaluation on the original and the transformed test set for the baseline CNNs with the highest obtained Top-1 accuracy reported.

We see that both the baseline 3D CNNs can classify the direction of the digits for **original**, but in the case of **treverse**, it fails, obtaining a Top-1 accuracy of 24.6% for the 4-layer Z3CNN and 24.13% for the Z3-ResNet-18.

We, therefore, see a significant difference between the Top-1 accuracies on the original test dataset versus the reversed dataset and can say that temporal filters in standard 3D CNNs are not equivariant to the temporal direction in videos and tend to learn strict temporal information.

4.5. Equivariance through data augmentation

To evaluate how data augmentation helps achieve equivariance to temporal direction in standard 3D CNNs, we train the two baseline 3D CNNs on the Moving MNIST-Dir dataset augmented with reversed videos. We train the 4-layer Z3CNN for 50 epochs, and the 3D-ResNet-18 for 100 epochs with a probability $p = 0.3$ of **treverse** during training time. We evaluate the Top-1 Accuracy on the original test set and a reversed test set. We use the same setup for the Moving MNIST dataset, augmented with random 90, 180 and 270-degree rotations digits, and obtain the baseline Top-1 accuracy for the original test dataset and test sets spatially rotated by 90, 180 and 270 degrees. The comparison of the baseline models trained with augmentation against the baseline models trained without augmentations can be found in Table 4 for augmentation with reversed videos during training, and in Table 5 for augmentations with spatial rotations.

We see that data augmentation with reversed videos significantly improves the performance of the baseline models on the Moving MNIST-Dir dataset(refer Table 4) while augmentation with spatial rotations significantly improves the performance of the baseline models on the Moving MNIST digits dataset(refer Table 5).

Model	Type	Top-1 Accuracy(%)	
		original	treverse
Z3CNN(no aug)	4-layer CNN	100	23.58
Z3CNN(aug)		100	99.87
T2CNN(no aug)		100	**99.93**
Z3-ResNet-18(no aug)	3D ResNet-18	100	24.6
Z3-ResNet-18(aug)		100	99.63
T2-ResNet-18(no aug)		100	**100**

Table 4: Comparison of Top-1 accuracy obtained on Moving MNIST-Dir dataset by the models with T2 convolutions against the baseline 3D CNN architectures containing standard 3D convolutions with and without augmentation with reversed videos. **aug** refers to augmentation with reversed videos applied during training, **original** refers to the untransformed test set, while **treverse** refers to the test set with temporal reverse transformations applied on it. We see that the models with T2 convolutions obtain significant improvement in accuracy compared to the baseline standard 3D CNN models with no augmentations applied while showing a small improvement in performance to the augmented baseline models.

4.6. Equivariance of Temporal direction using Temporal Equivariant 3D G-CNN

We analyse the benefit of replacing standard convolutions on 3D-CNNs with temporal equivariant filters by training the 4-layer T2CNN and the T2-3D ResNet-18 on the Moving MNIST-Dir dataset without any augmentations and evaluating the models on the test set for both **original** and **treverse**. The 4-layer T2CNN was trained for 50 epochs, while the T2-3D ResNet-18 was trained for 100 epochs. We compare the obtained Top-1 accuracies for both the untransformed and reversed test sets against the results obtained from the standard 3D CNNs, trained with and without augmentations. Table 4 summarises the results of the T2-CNNs for the untransformed test set and the reversed test set and shows the comparison between the standard 3D-CNNs.

We see that for the non-transformed test sets, the T2-CNNs achieve similar accuracy compared to the baseline CNNs trained with and without augmentations. On testing with reversed videos, we see a three-fold increase in the accuracy for the T2-CNNs compared to the baseline CNNs trained without data augmentation, with the 4-layer T2CNN reporting a Top-1 accuracy of 99.87% against the baseline CNN's 23.58%, while the T2-ResNet-18 reports a 100% Top-1 accuracy, against the baseline accuracy of 24.6% with standard convolutions. Compared to the baseline CNNs trained with augmentations, the models with T2 convolutions show a marginal increase in accuracy.

4.7. Equivariance of Spatial rotations using rotation equivariant 3D G-CNN

We also analyse the benefit of replacing standard convolutions with spatial rotation equivariant filters by training the 4-layer P43DCNN and the P43D-ResNet-18 on the Moving MNIST digits dataset with no augmentations. We then evaluate the models on the test set for original, r90, r180 and r270. We train the 4-layer P43DCNN for 100 epochs and the P43D-ResNet-18 for 150 epochs and compare the Top-1 accuracies for the original and the transformed test sets with the baseline CNNs with and without augmentation.

We see that for the non-transformed test sets, the models with P43D convolutions achieve marginally higher accuracy compared to the baseline models, with standard convolutions trained with and without augmentations. On testing with spatially rotated videos, we see a significant increase in the accuracy for the models with P43D convolutions compared to the baseline CNNs trained without data augmentation while showing a marginal increase in accuracy on videos rotated by 90 and 270 degrees and a marginal decrease in accuracy on videos rotated by 180 degrees.

4.8. Evaluating 3D-G-CNNs on HMDB-51 and UCF-101

HMDB-51 [12] was the first widely used video action recognition dataset. It consists of 6849 clips sourced from movies, YouTube and Google videos. The clips are divided into 51 action labels, with each label consisting of at least 100 videos. UCF-101 [11] was an extension of the existing UCF-50 dataset, and was introduced in 2012. It is a larger dataset than the HMDB-51 consisting of 13,320 videos sourced from YouTube. The videos are labelled under 101 action classes across a diverse set of actions, including variations in camera motion, object appearance, viewpoints, backgrounds and illuminations. Like HMDB-51, it contains 3 train and test splits.

We evaluate the four group equivariant models and the baseline CNNs on the first splits of the HMDB-51 and UCF-101 datasets. We train the 4-layer CNNs with an initial learning rate of 0.3, reduced by 10% at 100, 150 and 200 epochs, and training continues till 250 epochs for the HMDB-51. The 3D-ResNets were trained with an initial learning rate of 0.1, reducing by % at 100, 200 and 250 epochs and trained until 300 epochs. For the UCF-101, all networks were trained with an initial rate of 0.1, reducing by 10% at 50, 100 and 150 epochs, and training till 200 epochs. The training with both datasets for all models was done without any augmentations. Table 6 shows the top-1 accuracy results obtained on the UCF-101 and HMDB-51 using the first test split for the standard baseline CNNs and the Group equivariant CNNs.

On the UCF-101 dataset using the 4-layer CNN archi-

Model	Type	Top-1 Accuracy(%)			
		original	r90	r180	r270
Z3CNN(no aug)	4-layer CNN	99.67	8.33	25.17	9.82
Z3CNN(aug)		99.87	99.33	98.67	99.33
P43DCNN(no aug)		**99.93**	**99.93**	**99.67**	**99.87**
Z3-ResNet-18(no aug)	3D ResNet-18	99.59	8.47	23.66	8.81
Z3-ResNet-18(aug)		99.67	99.46	**99.77**	99.53
P43D-ResNet-18(no aug)		**99.83**	**99.67**	99.17	**99.63**

Table 5: Comparison of Top-1 accuracy obtained on Moving MNIST digits dataset by the models with P43D convolutions against the baseline 3D CNN architectures containing standard 3D convolutions with and without augmentation with rotations. **aug** refers to augmentations applied during training, **original** refers to the untransformed test set, while **r90**, **r180** and **r270** represent test sets with 90 degree, 180 degree and 270 degree spatial rotation transformations applied on them. We see that the models with P43D convolutions obtain significant improvement in accuracy compared to the baseline standard 3D CNN models with no augmentations applied, while showing similar performance to the augmented baseline model.

Architecture Type	Model	Top-1 Accuracy(%)	
		UCF-101	HMDB-51
4-layer CNN	Z3CNN(baseline)	36.83	20.86
	T2CNN	**37.35**	**21.58**
	P43DCNN	**37.03**	**21.99**
3D ResNet-18	Z3-ResNet-18(baseline)	50.86	22.03
	T2-ResNet-18	**51.44**	**23.59**
	P43D-ResNet-18	**51.38**	**23.56**

Table 6: Top-1 accuracy of 3D-G-CNNs compared against the baseline architectures on UCF-101 and HMDB-51. No augmentations were applied during training. We see that introducing group equivariant convolutions increases the accuracy for both the 4-layer CNN type architecture and the 3D-ResNet type architecture over the standard baseline architectures for UCF-101 and HMDB-51. Testing was performed on the first test split for all the networks.

tecture, the T2CNN shows a 1.41% improvement in Top-1 accuracy, while the P43DCNN shows an improvement of 0.54% in Top-1 accuracy over the baseline score obtained by the Z3CNN. For the 3D-ResNet type architectures, the T2-ResNet-18 shows an improvement of 1.1% and the P43D-ResNet-18 shows an improvement of 1.02% in the Top-1 accuracy over the standard 3D-ResNet-18.

On the HMDB-51 dataset, using the 4-layer CNN architecture, the T2CNN shows an improvement of $\%$ in the Top-1 accuracy, while the P43DCNN shows an improvement of 0.54% in Top-1 accuracy over the baseline score obtained by the Z3CNN. For the 3D-ResNet type architectures, the T2-ResNet-18 shows an improvement of 1.1% and the P43D-ResNet-18 shows an improvement of 1.02% in performance over the standard 3D-ResNet-18.

We also compare the Top-1 accuracy obtained by the models on transformed versions of the UCF-101 and HMDB-51 test datasets. The models with standard convolutions replaced with T2 convolutions are tested with **treverse** transformation applied to the test set, while the models with P43D convolutions are tested with **r90**, **r180** and **r270** transformations applied to the test set. Each test was carried out with a single spatial rotation applied during evaluation. The baseline models with standard 3D convolutions are tested on all transformations, and we compare the results of the temporal reverse transformation in Table 7, and the spatial rotations transformations in Table 8.

Model	Type	Top-1 Accuracy(%)	
		UCF-101	HMDB-51
Z3CNN	4-layer CNN	31.45	20.54
T2CNN		**37.07**	**22.07**
Z3-ResNet-18	3D ResNet-18	47.55	21.71
T2-ResNet-18		**51.07**	**23.53**

Table 7: Results of testing with UCF-101 and HMDB-51 test sets consisting of reversed videos. The **treverse** transformation is applied during test time. Compared to the baseline 3D CNNs with standard convolutions, the models with T2 convolutions show increased Top-1 accuracy.

From the results, we observe that for the models with T2 convolution, there is an increase in the performance over the baseline CNN with standard convolutions for reversed videos on both datasets, and the P43DCNN achieves significantly improved performance on all spatially rotated videos for both datasets as well.

5. Conclusion

We have discussed two symmetry groups for video action recognition. We introduced the temporal direction equivariant group T2 and extended the p4 group to 3D to make the spatial rotation equivariant group P43D.

We also introduce the Moving MNIST-Dir dataset consisting of handwritten MNIST digits moving over a black

Model	Type	Top-1 Accuracy(%)					
		UCF-101			HMDB-51		
		r90	r180	r270	r90	r180	r270
Z3CNN	4-layer CNN	8.71	12.65	8.36	8.71	12.65	8.36
P43DCNN		**36.06**	**34.02**	**34.58**	**20.78**	**20.59**	**21.27**
Z3-ResNet-18	3D ResNet-18	10.55	15.75	10.15	9.93	9.41	9.74
P43D-ResNet-18		**21.07**	**26.65**	**22.2**	**16.99**	**16.54**	**17.32**

Table 8: Results of testing with UCF-101 and HMDB-51 test sets consisting of videos transformed with spatial rotations. r90 corresponds to a spatial rotation of 90 degrees CCW, r180 corresponds to a spatial rotation of 180 degrees CCW and r270 corresponds to a spatial rotation of 270 CCW about the origin. All transformations are applied only during test time, with individual tests carried out for each transformed test set. Compared to the baseline 3D CNNs with standard convolutions, the models with P43D convolutions show increased Top-1 accuracy across all rotated orientations.

background distributed over four classes representing the directional plane of movement.

Our results show that standard 3D CNNs are not equivariant to reversed videos and spatial rotations when trained on unaugmented datasets. Replacing the standard convolutions with the T2 group convolutions in 3D CNNs improves the accuracy of the network to reversed videos, even when trained without augmentations. We see a similar result when replacing standard convolutions with P43D group convolutions on transformations that leave the label unchanged.

We also see an increased accuracy for the group equivariant CNNs on untransformed small video action recognition datasets UCF-101 and HMDB-51, compared to the baseline standard 3D CNNs suggesting that 3D G-convolutions improve the performance of networks on datasets where actions are typically unidirectional, and objects are upright in the frame. We also see improved performance on the UCF-101 and HMDB-51 for rotated videos compared to the baseline.

The main limitation of this paper is that it assumes label preservation of videos on transformation. However, changing the temporal direction or spatially rotating frames in videos often leads to the action portrayed in the frames being different from the label associated with it. For example, reversing a video where the action is *Placing an object on a table* leads to the action becoming *Picking up an object from the table*. The network would classify the reversed video with the first label in the current implementation.

References

[1] T. S. Cohen and M. Welling, "Group Equivariant Convolutional Networks," *arXiv:1602.07576 [cs, stat]*, Jun. 2016, arXiv: 1602.07576. [Online]. Available: http://arxiv.org/abs/1602.07576

[2] E. Hoogeboom, J. W. T. Peters, T. Cohen, and M. Welling, "Hexaconv," *ArXiv*, vol. abs/1803.02108, 2018.

[3] T. S. Cohen and M. Welling, "Steerable cnns," 2016.

[4] M. Weiler, F. A. Hamprecht, and M. Storath, "Learning steerable filters for rotation equivariant cnns."

[5] M. Winkels and T. S. Cohen, "3D Group-Equivariant Neural Networks for Octahedral and Square Prism Symmetry Groups," p. 5.

[6] T. S. Cohen, M. Geiger, J. Koehler, and M. Welling, "Spherical cnns," 2018.

[7] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen, "3d steerable cnns: Learning rotationally equivariant features in volumetric data." [Online]. Available: https://github.com/mariogeiger/se3cnn.

[8] J. Linmans, J. Winkens, B. S. Veeling, T. S. Cohen, and M. Welling, "Sample efficient semantic segmentation using rotation equivariant convolutional networks," 2018.

[9] B. Chidester, T.-V. Ton, M.-T. Tran, J. Ma, and M. N. Do, "Enhanced rotation-equivariant u-net for nuclear segmentation." [Online]. Available: https://github.com/thatvinhton/G-U-Net.

[10] D. K. Gupta, D. Arya, and E. Gavves, "Rotation equivariant siamese networks for tracking," 2020.

[11] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild," *arXiv:1212.0402 [cs]*, Dec. 2012, arXiv: 1212.0402. [Online]. Available: http://arxiv.org/abs/1212.0402

[12] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb51: A large video database for human motion recognition," 11 2011, pp. 2556–2563.

[13] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," *arXiv:1705.07750 [cs]*, Feb. 2018, arXiv: 1705.07750. [Online]. Available: http://arxiv.org/abs/1705.07750

[14] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," 2015.

[15] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 6450–6459. [Online]. Available: https://ieeexplore.ieee.org/document/8578773/

[17] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan 2013.

[18] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[19] D. Wei, J. J. Lim, A. Zisserman, and W. T. Freeman, "Learning and Using the Arrow of Time," p. 9.

[20] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman, "The Kinetics Human Action Video Dataset," *arXiv:1705.06950 [cs]*, May 2017, arXiv: 1705.06950. [Online]. Available: http://arxiv.org/abs/1705.06950

[21] L. Sevilla-Lara, S. Zha, Z. Yan, V. Goswami, M. Feiszli, and L. Torresani, "Only Time Can Tell: Discovering Temporal Data for Temporal Modeling," *arXiv:1907.08340 [cs]*, Oct. 2019, arXiv: 1907.08340. [Online]. Available: http://arxiv.org/abs/1907.08340

[22] W. Price and D. Damen, "Retro-Actions: Learning 'Close' by Time-Reversing 'Open' Videos," p. 10.

[23] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, "The Jester Dataset: A Large-Scale Video Dataset of Human Gestures," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 2874–2882. [Online]. Available: https://ieeexplore.ieee.org/document/9022297/

[24] R. Goyal, S. E. Kahou, V. Michalski, J. Materzyńska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax, and R. Memisevic, "The "something something" video database for learning and evaluating visual common sense," *arXiv:1706.04261 [cs]*, Jun. 2017, arXiv: 1706.04261 version: 2. [Online]. Available: http://arxiv.org/abs/1706.04261

[25] "Mnist handwritten digit database, yann lecun, corinna cortes and chris burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/

2

Basics of Deep Learning

Deep learning is a field under the umbrella of machine learning which relies on the use of structured artificial neural networks [1] to learn representations from input data such as images, videos, audio, or text. Unlike conventional machine learning algorithms, deep learning models automatically learn which features are to be extracted from the input data for a particular task. The training process in deep learning can be either supervised, semi-supervised or unsupervised. The most commonly used training method is supervised, where networks learn from a labelled training dataset.

2.1. Feed-forward Networks

A feed-forward network, as the name suggests, is a type of artificial neural network in which information flow is unidirectional - from the input data x, through successive intermediate computations to reach an output y, without any feedback connections, cycles or loops [2]. Feed-forward networks approximate a function $y = f^*(x)$, mapping the input x to an output y. In the case of a classifier, the input x is mapped to a class label y. A feed-forward network thereby defines a mapping $y = f(x; \theta)$ and learns the values of the parameters in θ to find that result that is the best approximation of the function on a given training set. Since the input layer and the output layer are connected through a number of intermediate layers, these networks are also called *Multi-layer Perceptrons.*

Feed-forward networks perform computations by a composition of different functions, often represented as a Directed Acyclic Graph, which describes how the functions are composed together in the form of a chain. The length of this chain determines the depth of the network. The last layer of the network is called the output layer, while the layers in between are called hidden layers. Figure 2.1 shows a multi-layer perceptron with a depth of 3, consisting of an input layer, an output and one hidden layer. Figure 2.2 represents another network with a depth of 5, consisting of three hidden layers.

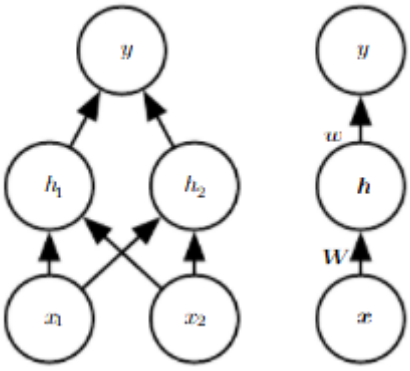


Figure 2.1: A multi-layered perceptron with a depth of 3, represented as a directed acyclic graph. The learnable parameters are represented by the vector W for the mapping between x and h, and the vector w for the mapping between h and y. Image taken from [**?**]

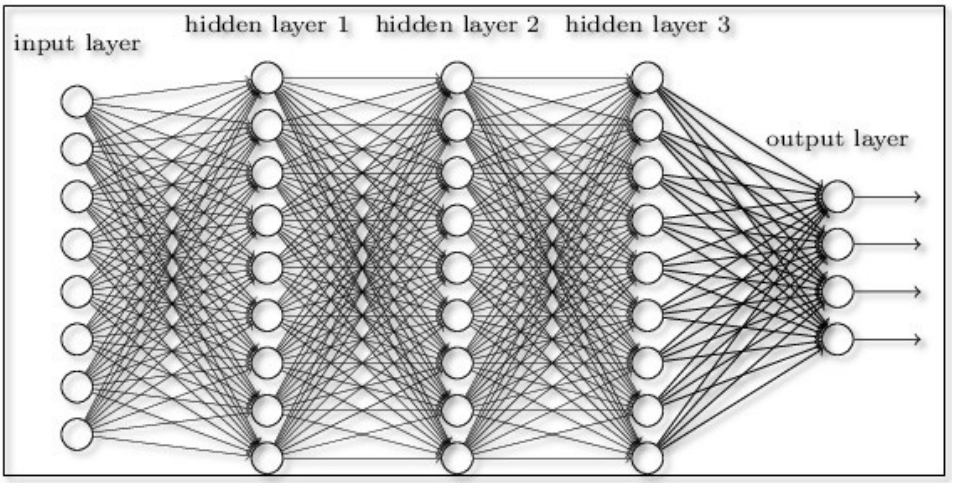


Figure 2.2: A multi-layered perceptron with a depth of 5, consisting of 3 hidden layers. Image taken from [3]

2.1.1. Activation function

The output of an MLP with a single hidden layer is denoted by the function $f(x) = \omega^T g(x, W^T) + b$, where b is called the bias term. If the function g, which computes the output of the hidden layer, is a linear function, then $f(x)$ would be a linear function as well, which limits the ability of the MLP to approximate non-linear functions such as XOR. We overcome this limitation by adding a level of complexity to the outputs of the hidden layers using non-linear activation functions. [4, 5, 6]

For multi-class classification tasks, the most commonly used function is the Rectified Linear Unit(ReLU) [4]. The ReLU function simply clips any negative outputs to 0. The function can be represented as:

$$g(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

This, however, leads to problems with negative inputs, as the function renders neurons inactive, thereby affecting the optimisation process. The problem is addressed by a modified function called the Leaky ReLU [6], which replaces the zero for negative inputs with a small coefficient. The Leaky ReLU function can be represented as:

$$g(x) = \begin{cases} 0.01x & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

Figure 2.3 shows the plots of the ReLU and LeakyReLU activation functions with input in the range $[-10, 10]$ with a coefficient of 0.05 for negative inputs in LeakyReLU

2.2. Optimization

During backpropagation or the backward pass, optimisation usually involves minimising or maximising a loss or cost function. In most classification tasks where the objective is to predict a label based on an input, the cost function is a cross-entropy between the predicted output of the network and the true label associated with the input.

In the case of multi-class classification consisting of N target classes, the label vector is encoded as a one-hot vector.

$$t = (0, 0...., 1, 0, 0...0) \tag{2.1}$$

where the nth entry is 1, representing the label associated with a particular input. A **softmax function** is used to map the output to an N-dimensional, non-negative output vector whose sum adds up to 1. Thus, the output vector represents a probability distribution over the N classes. The cross-entropy loss function for multi-class classification is therefore defined as

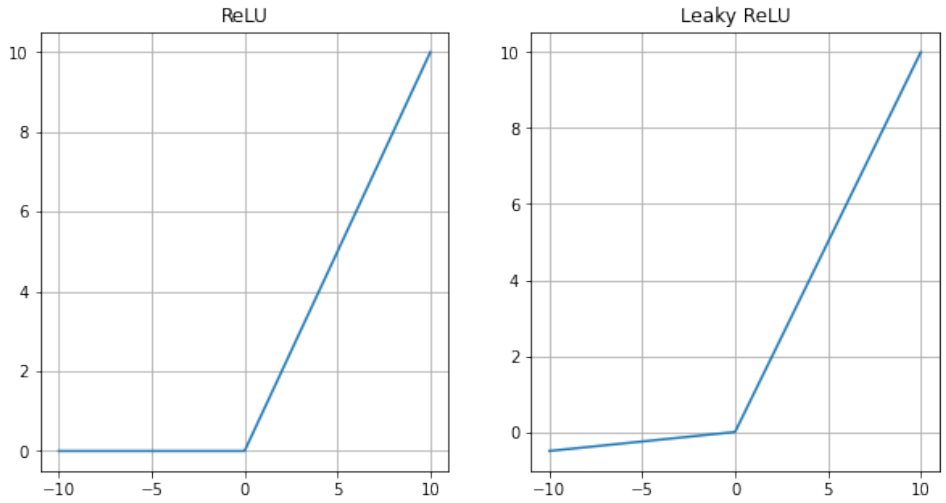


Figure 2.3: Plots of Rectified Linear Unit(ReLU) and Leaky ReLU. The plots are obtained using an input $x \epsilon [-10, 10]$ and a coefficient of 0.05 for the LeakyReLU activation function.

$$\mathcal{L}_{CE}(y,t) = \sum_{n=1}^{N} t_k \, log \, y_k$$
$$= -t^T (log \, y)$$

where y and t represent the predicted labels and the true labels, respectively

2.2.1. Stochastic Gradient Descent

Gradient descent is one of the most common optimization algorithms in the field of machine learning. Gradient descent minimizes a function $f(x)$ making small movements opposite to the sign of the derivative $f'(x)$. Let us consider a function $f(x) = \frac{1}{2}x^2$, and its first-order derivative to be $f'(x) = x$. The function has a global minima at $x = 0$, when $f'(x) = 0$. Figure 2.4 illustrates the function, its derivative and its local minima. In the case of neural networks, the function is defined as $y = f(x, \theta)$ where x represents the input and θ represents the weights. The gradient of the function is defined as a partial derivative with respect to the weights

$$\nabla_\theta f(x,\theta) = \frac{\partial}{\partial \theta_i} f(x,\theta)$$

The weights are updated by gradient descent, taking a step of size ϵ can be written as

$$\theta' = \theta - \epsilon \nabla_\theta f(x,\theta)$$

In deep learning terminology, the step size ϵ is called the *learning rate*, and the gradient is approximated on the cost function over the training samples.

Stochastic Gradient Descent(SGD) is an extension of the gradient descent algorithm, and is one of the most widely used optimization algorithms in deep learning. The main difference between SGD and gradient descent, lies in the fact that SGD views the gradient as an expectation, which can be estimated using a mini batch of samples. On each optimization step, a mini batch $B = \{x^{(1)}, ...x^{(b)}\}$ drawn i.i.d from the training set. The mini-batch size b is typically very small compared to the size of the training set. and the gradient of the cost function is approximated using these small number of samples. For a set of b' samples, the SGD update formula can be written as

$$\theta' = \theta - \epsilon \frac{1}{b'} \sum\nolimits_{i=1}^{b'} \nabla_\theta (x_i, y_i, \theta) \tag{2.2}$$

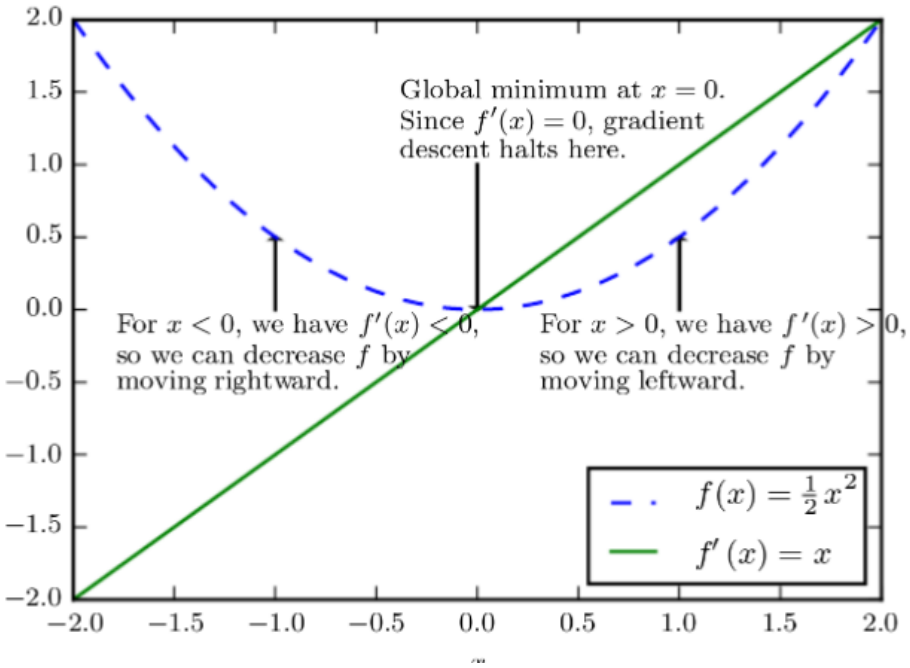


Figure 2.4: Gradient descent algorithm showing the global minimum of the function $f(x) = \frac{1}{2}x^2$ [**?**]

2.2.2. Learning rate decay

In deep learning, learning rate schedules aim to adjust the learning rate of the network during the training process by reducing the learning rate value by a pre-defined factor according to a pre-defined schedule. Altering the learning rate of an optimisation technique such as stochastic gradient descent can improve performance while cutting down on the training time. Figure 2.5 depicts the two scenarios of training a network - one with a constant learning rate and the other using learning rate decay. In the figure 2.5a, a constant learning rate is used, and we see that the optimisation steps, represented by the blue line, lead to a very noisy convergence, and in several iterations, deviates from the minimum. In the figure 2.5b, the green line represents an optimisation process with learning rate decay. We see that the optimisation starts with a higher learning rate value, and as it moves towards the minimum, the learning rate gets smaller, and the oscillations about the minimum become closer and lead to faster convergence.

Step Decay

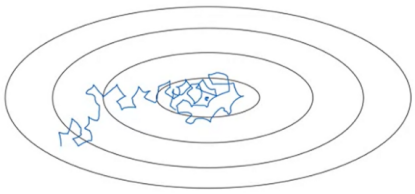
Step decay is a learning rate schedule where the learning rate is reduced a factor every few epochs, where the number of epochs can be considered as a hyperparameter. The learning rate is calculated using step decay by the following equation:

$$LR^1 = LR^0 * \alpha^{\lfloor epoch/h \rfloor}$$

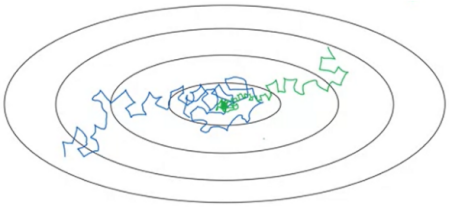
where α represents the factor by which the learning rate is reduced, and h represents the number of epochs when the learning rate is updated. A visual representation of the step learning rate decay is shown in figure 2.6.

2.3. Convolutional Neural Networks

Convolutional neural networks, or CNNs, [9] is a type of deep learning network that is regarded as the standard for performing computer vision tasks that deal with structured arrays of data - such as classifying images [10, 11], classifying actions in videos [12, 13, 14], detecting objects [15], segmentation [16, 17], and even generating images and videos [18]. Convolutional networks are neural networks which use the convolution operation in at least one of its layers. [2].



(a) Optimization with a constant learning rate leading to a noisy convergence.



(b) Optimization with a decayed learning rate (green) compared to constant learning rate (blue)

Figure 2.5: Comparing optimizations with constant learning rate and decayed learning rate. Figure sourced from [7]

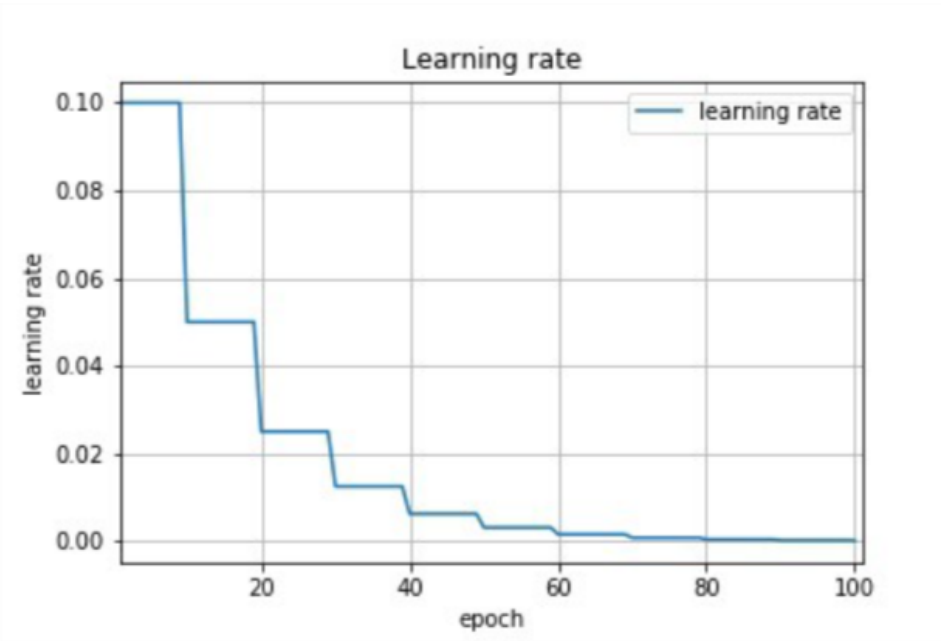


Figure 2.6: Learning rate reduction using step decay. Image sourced from [8]

A typical CNN consists of multiple convolutional layers with activation functions and pooling layers in between. Figure 2.7 shows a CNN that classifies handwritten digits. The input to the pipeline is an image containing a handwritten digit. The convolutional layers learn the features, and the classification is done using fully connected layers; and finally, a softmax activation provides a vector of probabilities with a size equal to the number of classes (10 in this case).

2.3.1. Convolution operation

Convolution is the base operation that is performed within CNNs and consists of three main components - the input, the kernel function, and the resultant feature map. In machine learning, the input is usually a multidimensional array of data, such as an image or a video, and the kernel is a multidimensional array

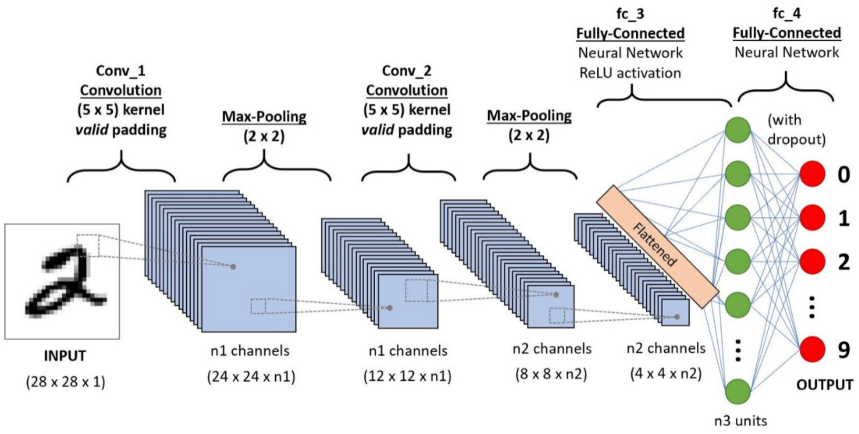


Figure 2.7: Convolutional neural network to classify handwritten digits. [9]

of weights that are updated and optimised by the learning algorithm. Interestingly, the dimensionality for both the input, the output, and the kernel is the same for the convolution operation. Thus, for a two-dimensional input I, the kernel K would also be two-dimensional, and the resultant feature map S would also be two-dimensional. The convolution operation is represented by $*$ and is defined by the equation 2.5

$$S(i,j) = (K \star I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{2.3}$$

It may be interesting to note from equation 2.5 that the kernel is flipped relative to the input, i.e. when m and n increase, the index of the kernel increases, but the index of th input decreases. This leads to convolutions being commutative, i.e.

$$(K \star I)(i,j) = (I \star K)(i,j) \tag{2.4}$$

For most libraries implementing convolutions however, the commutative property does not hold a very high importance, and convolution is instead implemented as a related function called cross-correlation, which is the same as convolution but without the kernel being flipped relative to the image.

$$S(i,j) = (K \star I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.5}$$

Figure 2.8 shows the example of a 2D convolution without kernels being flipped. In the figure, the 2x2 kernel slides over a 4x3 image, and the weighted sum is calculated using the kernel weights and the pixels from the area of the image under the kernel. The resultant feature map is of the size 3x2.

2.3.2. Feature detection using kernels

In convolution networks, kernels are referred to as filters, and a convolutional layer is characterised by the size of the kernels, the number of filters and the stride. The weights or parameters associated with a kernel act as feature detectors, and hence, during training and optimisation, learning the weights of the kernels translates to learning the features of the input. To demonstrate how kernels detect features from the input, we refer to figure 2.10, where two different kernels are used to detect features in two images. In the first image, a round kernel is used to detect the round shape of the sunflowers. We see that the feature is detected by looking at the highlighted portions on the resultant activation map. The second image is a still from the popular game *Where's Waldo*, and to detect the character within the image, a kernel representing the shirt of the character is used, and the resultant activation map highlights the location of the character.

2.3.3. Pooling

Pooling functions are the final step in a convolution pipeline and work by replacing the network's output at a particular location with an aggregated value using the nearby inputs. To take an example, a **max**

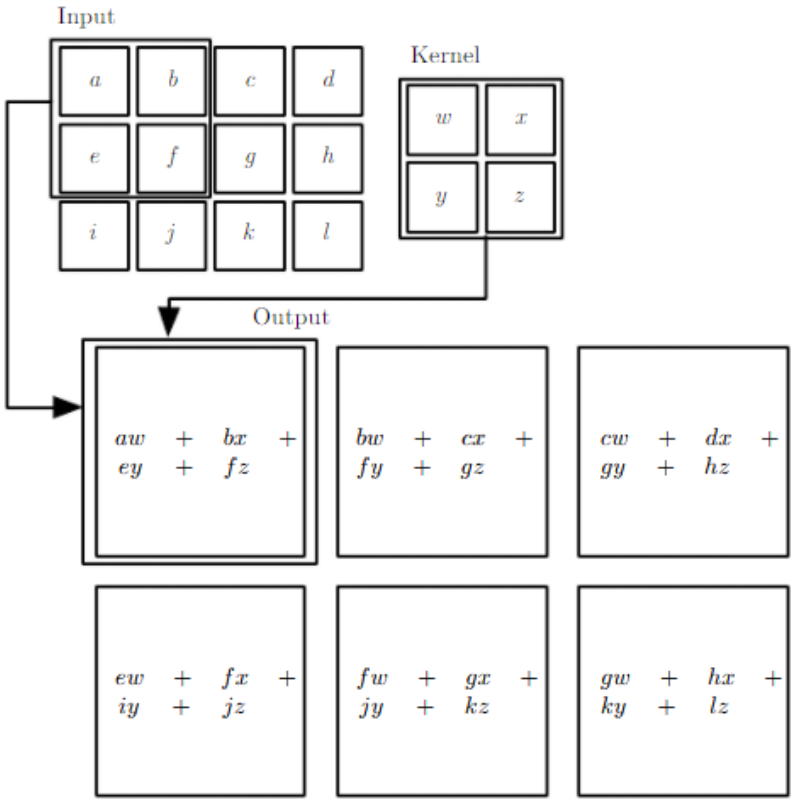


Figure 2.8: Simple example of a 2D Convolution performed on an image of size 4x3 using a kernel of size 2x2 with a stride of 1. [2]

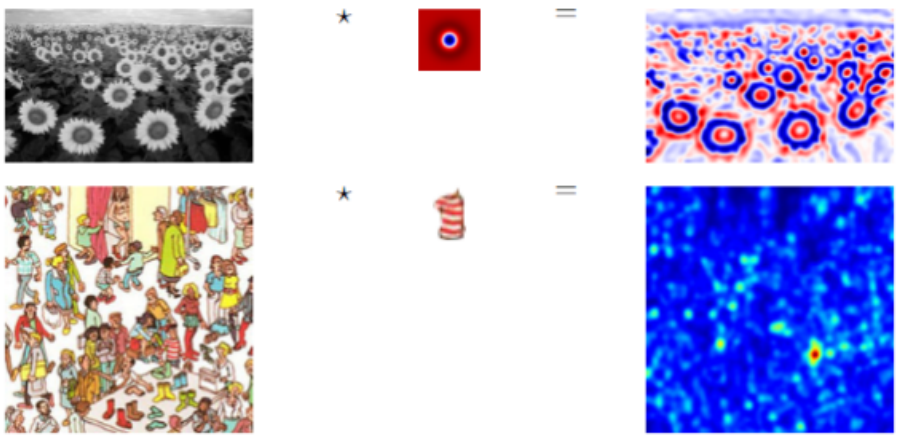


Figure 2.9: Convolution kernels being applied to images and the resultant activation maps highlighting the detected features. [**?**]

pooling [**?**] operation would produce a value that is the maximum output within the neighbourhood of the pooling kernel. Other commonly used pooling functions include average pooling, L2 norm and weighted averages from the central pixel. Pooling is beneficial as it helps make the feature representation approximately invariant to small input translations. What this means is that even if the input is translated by a small amount, the resultant values obtained from pooling does not change. Invariance to translation is an advantageous property if the task is more concerned about the presence of a particular feature instead of its exact position. Figure **??** shows a set of max pooling detectors on two images of the handwritten digits and the activated detector unit for the specific input with the most significant

response.

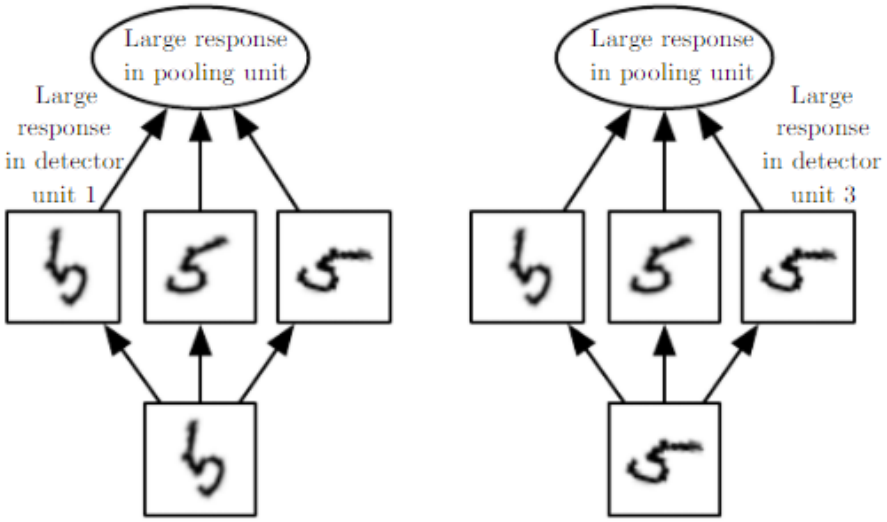


Figure 2.10: Max pooling on two different images of the handwritten digit 5, showing activated detector units. [2]

Figure 2.11 presents the complete overview of a CNN layer pipeline consisting of 3 filters. We start with the input image of a windmill and perform the convolution operation of the three kernels in the convolution block. It is then followed by a non-linear activation(ReLU) in the non-linearity layer and finally is pooled in the Spatial pooling layer to produce the output that is fed into the next convolutional layer.

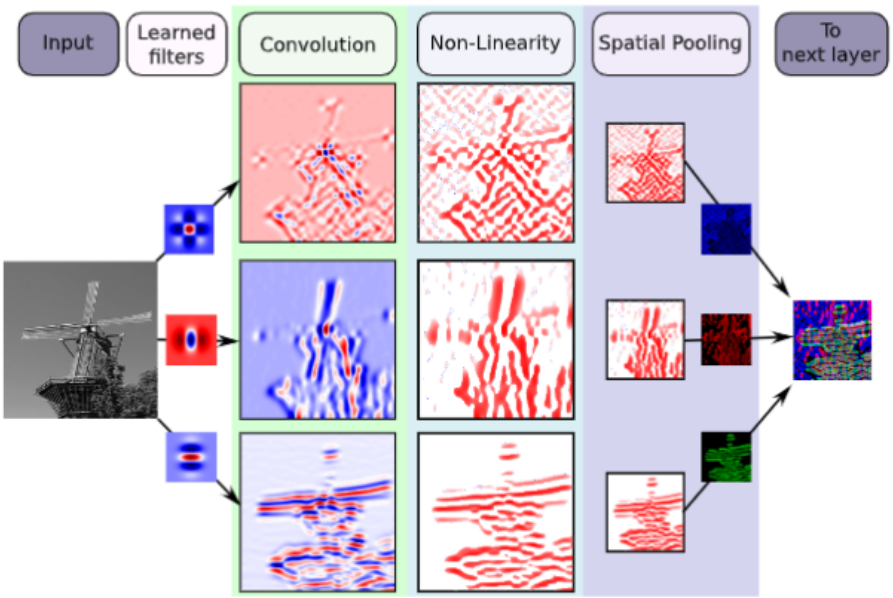


Figure 2.11: An overview of a convolution layer containing a convolution block with three filters, a ReLU non-linear activation block and a spatial pooling block. [**?**]

3

Action Recognition

Action recognition is one of the representative tasks in computer vision and is defined as identifying a particular activity being performed in an image or video by a human subject. Despite the staggering success of image classification using deep neural networks, the progress in learning video representations has been significantly slower. This can be attributed to some challenges inherent in the task.

- **Computational cost:** There is a significant computational cost involved in action recognition from videos, as the same 2D architecture, when inflated to 3D, results in a great increase in the number of trainable parameters, which leads to longer training time and requires more resources.
- **Capturing long-range context:** While action recognition from an image is a case of pose and object detection, action recognition in a video, the network should be able to hold the context of consecutive video frames and use this state information to predict a task. Moreover, temporal information needs to be robust to changes in camera position and background.
- **Variety of options:** There are multiple options available when designing deep learning network architectures that aim to capture the spatio-temporal information, such as using RGB frames, or using optical flow [19], or a combination of both. The choice is non-trivial and makes networks expensive to evaluate.

3.1. Action recognition Datasets

3.1.1. HMDB-51

Introduced in 2011, HMDB-51 [20] was the first widely used video action recognition dataset. It consists of 6849 clips sourced from movies, YouTube and Google videos. The clips are divided into 51 action labels, with each label consisting of at least 100 videos.

3.1.2. UCF-101

UCF-101 [21] was an extension of the existing UCF-50 dataset, and was introduced in 2012. It is a larger dataset compared to the HMDB-51 consisting of 13,320 videos sourced from YouTube. The videos are labelled under 101 action classes across a diverse set of actions, including variations in camera motion, object appearance, viewpoints, backgrounds and illuminations. Like HMDB-51, it contains 3 train and test splits.

3.1.3. Kinetics

Kinetics is currently the most widely accepted video action recognition benchmark. To put it simply, Kinetics is to action classification what ImageNet is to image classification. The Kinetics-400 dataset [13], introduced in 2017, consists of nearly 280k videos split into 400 human action labels. More versions of the Kinetics dataset were recently introduced containing even more data and labels, with Kinetics-600 consisting of 480k videos in 600 classes and the Kinetics-700 consisting of 650k videos in 700 classes.

3.1.4. Moments in time
Moments in time [22] is one of the latest large-scale datasets for action recognition consisting of nearly 1 million videos distributed within 339 classes. Unlike other datasets that focus on human actions, Moments in time consists of videos with animals, objects, natural phenomena, and videos involving humans.

3.1.5. Something-Something
The Something-Something dataset [23] was first introduced in 2017 and consists of 174 action classes that depict simple human actions performed with simple objects, which humans perform daily. The latest version of the dataset consists of over 220k videos. This dataset is another popular benchmark as it requires good temporal modelling as most of the actions in the dataset can not be inferred from spatial features alone.

3.2. Common architectures for Action Recognition
Learning representations from videos requires networks to be able to extract both spatial as well as temporal features from the provided input data. This challenge was addressed by either using stacked RGB frames [12, 14, 24], or using a combination of RGB frames and optical flow input [13, 25]. This section provides an overview of the different methods used to learn representations from videos.

3.2.1. Video Representations using Long Short-Term Memory Networks
Ullah et al. [26] used LSTMs for representing videos but with the addition of Convolutional Neural Network(CNN). This model analyzes frame by frame action videos. A CNN is used to extract deep features out of every sixth video frame, and these features are then fed into an LSTM. The LSTM processes the video in five frames, and the output for small chunks is combined for the final output. As we can see from figure 3.2, if a video has K frames, it will be processed in K time steps, thus making this representation suitable for long, complex videos. The results of this model on the UCF-101 dataset is shown in figure 3.3.

3.2.2. Optical Flow
Horn and Schunck [27] defined optical flow as the distribution of apparent velocities of movement of brightness patterns in an image. Optical flow can also be defined as the pattern of apparent motion of the contents of a visual scene, such as objects, edges, and surfaces brought about by the relative motion between the observed content and the observer. Due to this, optical flow provides crucial information on the arrangement of observed objects in the spatial domain and the rate of change of the arrangement. Beauchemin and Barron [28] describes optical flow as an approximation of image motion as a projection of velocities of three-dimensional surface points onto the imaging plane of a visual sensor. As the surface points progress through a temporal domain, these displacements or instantaneous image velocities projected onto a two-dimensional plane allow motion estimation.

Figure 3.4a shows the image of the Rubik's cube at a time instance t. At time $t + \delta t$, the image changes to the figure 3.4b. Figure 3.4c shows the optical flow field denoted by the arrows, which show the direction of the estimated movement of the surface points on the projected image.

3.2.3. Two-Stream CNNs
Simonyan and Zisserman [30] stated that videos could be naturally decomposed into spatial and temporal components. The spatial portion of the video is represented by a set of individual frames and carries information about the scenes and objects depicted in the video. The temporal portion conveys the motion across the frames or the movement of the observer or objects over time. Therefore, two-stream Networks similarly perform video recognition by splitting the architecture into the spatial and temporal components and processing them in parallel using 2D or 3D Convolutional Networks. In [30], each stream is implemented using a deep ConvNet, softmax scores of which are combined by late fusion. Figure 3.5 describes the two-stream architecture used for video classification by Simonyan and Zisserman [30].

Inflated 3D ConvNet(I3D) [13] modified the two-stream architecture further by replacing the 2D spatial and temporal convolution networks with 3D Convolutions. The results from the two streams are

Figure 3.1: Examples from popular video action datasets. At the top we have single frames from different classes from the UCF-101 dataset, followed by single frames from classes in the HMDB-51 dataset, and the Kinetics dataset respectively, and at the bottom we have consequtive frames from the Something-Something dataset for two classes.

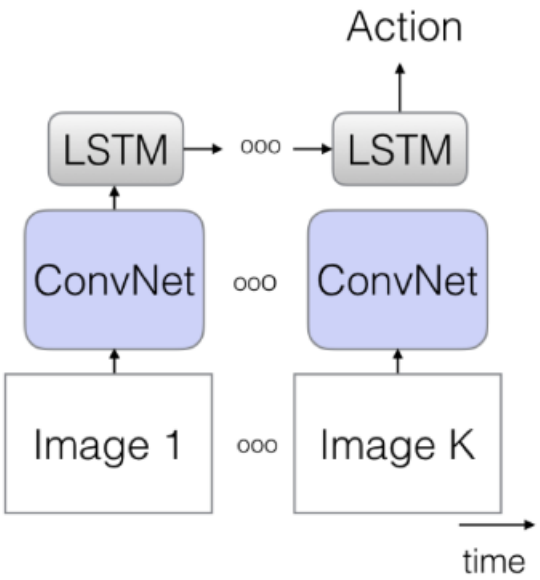


Figure 3.2: Video representation using CNN and LSTMs

averaged as the final step, and the predicted label is the output. I3D was able to adapt proven image classification architectures to use in the 3D CNNs, and also initialize optical flow networks [**?**] to inflate ImageNet pre-trained 2D weights to 3D, subverting the need to train the models from scratch. With pre-training on Kinetics-400, I3D achieved state-of-the-art performance on both UCF-101 and HMDB-51. It also paved the way for a standardized benchmark for video action recognition, requiring subsequent publications to report performance on Kinetics-400.

Spatial Stream Network
The spatial stream network operates on RGB frames performing action recognition and classification from still images. Actions associated with objects can be recognized easily as they provide crucial clues from the static images.

Motion Stream Network
The optical flow network operates on the temporal domain of the video. Unlike the input to the spatial domain, the input to the Optical flow ConvNet is formed by stacking optical flow displacement fields between several consecutive frames. The input thereby describes the motion between video frames, which makes the recognition easier, as the network does not need to estimate motion implicitly [30]. Figure 3.6 shows the optical flow, where the (a) and (b) show two consecutive frames, (c) shows the optical flow field of the action, (d) shows the horizontal component of the displacement vector, and (e) shows the vertical component of the displacement vector.

Fusing the Streams
Figure 3.7 shows the various methods of fusing the two parallel streams in increasing order of performance as well as the number of trainable parameters, from left to right. In Figure 3.7a we see the spatial stream with a single RGB Frame, which is enough to recognize an action based on the scene and objects, and the motion stream with the optical flow over time, which are merged and used to recognize the output. Figure 3.7b shows a single RGB Frame again as the input to the Spatial ConvNet and the motion from Optical flow frames as the input to the Optical flow ConvNet and also spread across several feature maps. Finally, the feature maps are again fed into a 3D ConvNet whose output is the action performed in the video. Finally, in Figure 3.7c we see Two Stream 3D Convolutional Networks which process a number of RGB Frames as input to the Spatial 3D Conv Net, and the optical flow frames depicting the motion in the Optical Flow 3D ConvNet, and the results are merged in the end and used to predict the action labels.

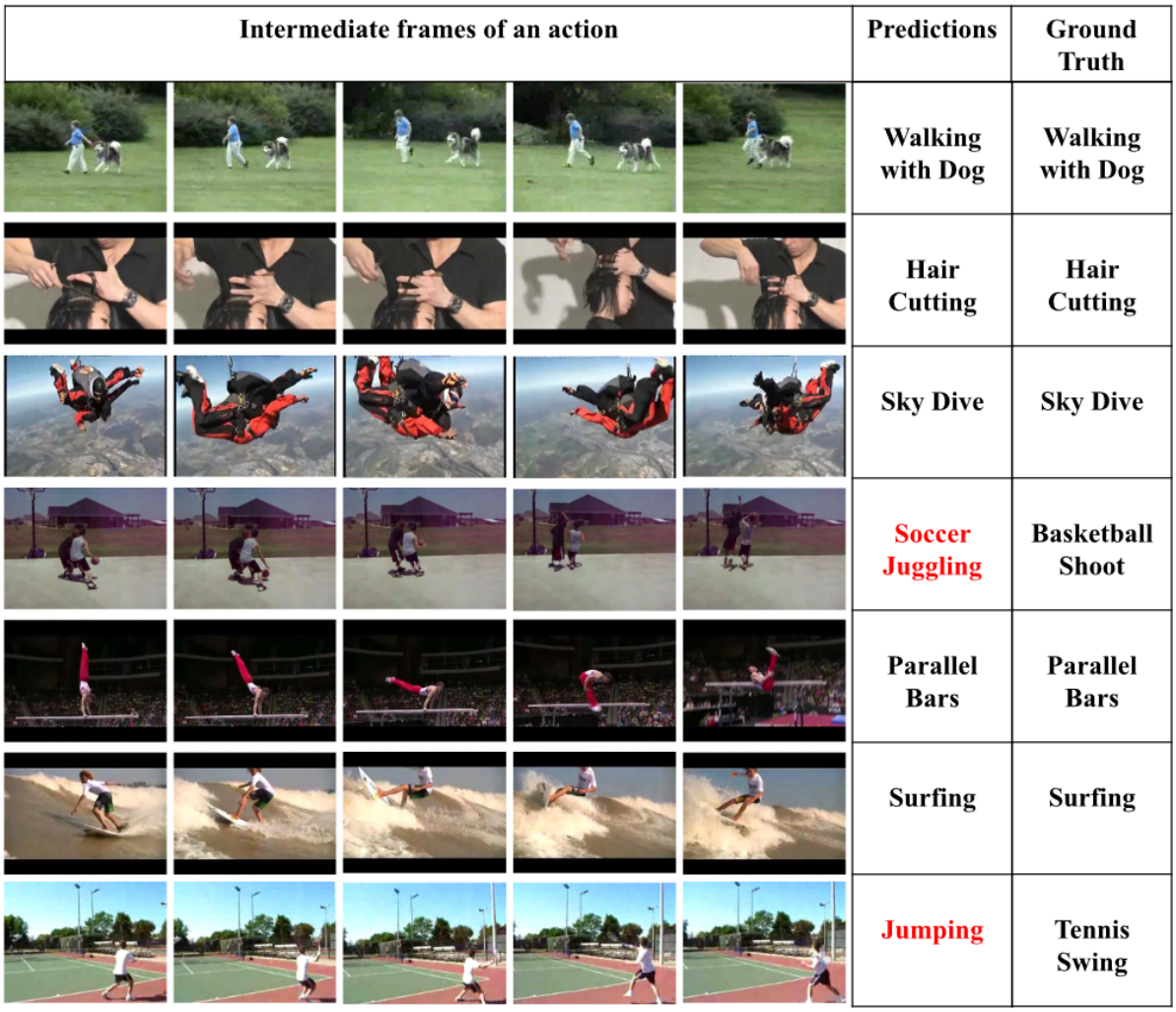
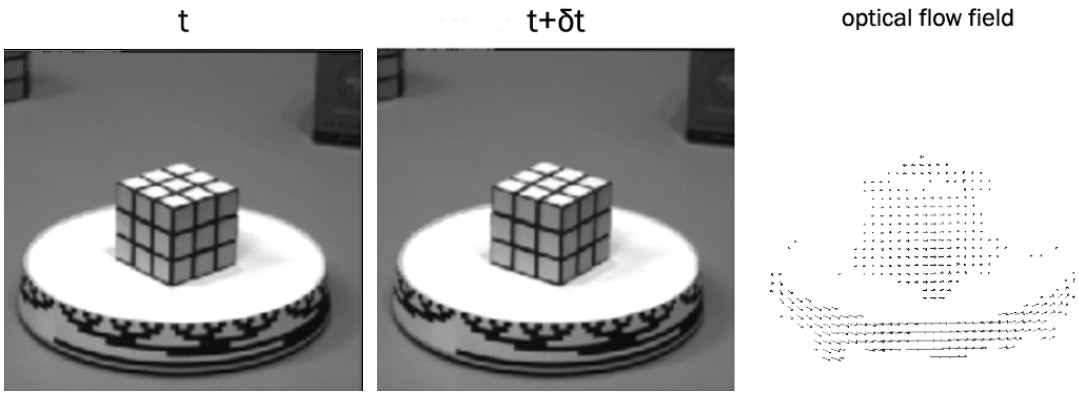


Figure 3.3: Predictions of LSTM video representation method for action recognition for sample clips. Wrong classifications are marked in red.[26]



(a) Image at time = t **(b)** Image at time = t + δ t **(c)** Optical flow field

Figure 3.4: Depicting optical flow field of the image of the Rubik's Cube between two temporal instances [29]

3.2.4. 3D Convolutional Networks

A 3D Convolution is an extension over the 2D spatial convolution, which has three-dimensional kernels that convolve along the spatial and temporal dimensions. 3D Convolution Kernels are used to build

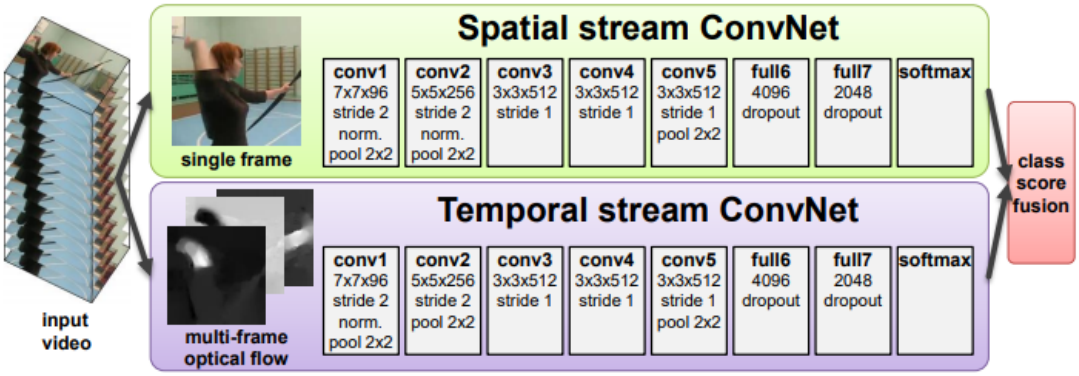


Figure 3.5: Two-stream architecture for video classification [30]

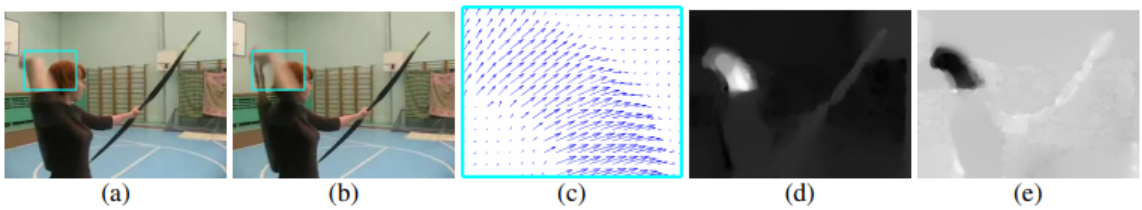
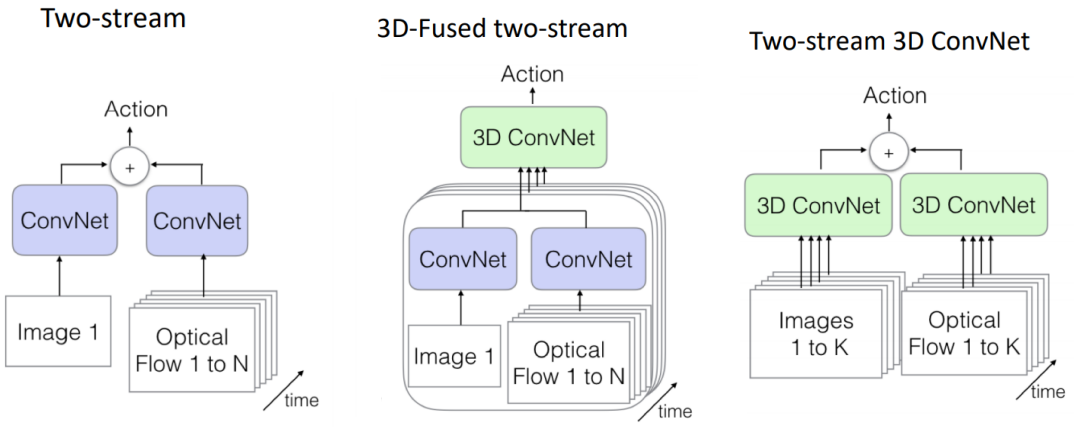


Figure 3.6: Optical flow of actions performed in a video in temporal domain [30]



(a) Two-Stream Network **(b)** 3D-Fused Two-Stream Network **(c)** Two-Stream 3D Convolutional Network

Figure 3.7: Methods of fusing the parallel streams in Two-Stream Networks. Image sourced from [13]

3D-CNNs such as Res3D [12], C3D[14], and I3D[13]. These state-of-the-art video learning models use multiple three-dimensional convolutional layers to learn robust video representations with high accuracy but with a high overhead as well. A 3D CNN takes a spatio-temporal volume as the input, such as a video and using the 3D Kernel convolves over the spatio-temporal domain. The motion information is captured by convolving over the temporal dimension. [31]. Figure 3.8 shows how a 3D CNN captures features along the spatio-temporal domain.

C3D

Tran et al. [14] proposed an effective approach for learning spatio-temporal features from videos by using deep 3D Convolutional Networks built using three-dimensional kernels. Through the paper, the authors show that 3D Convolutional Networks perform well as feature learning machines that can simultaneously model both spatial and temporal features(appearance and motion) and that 3D CNNs

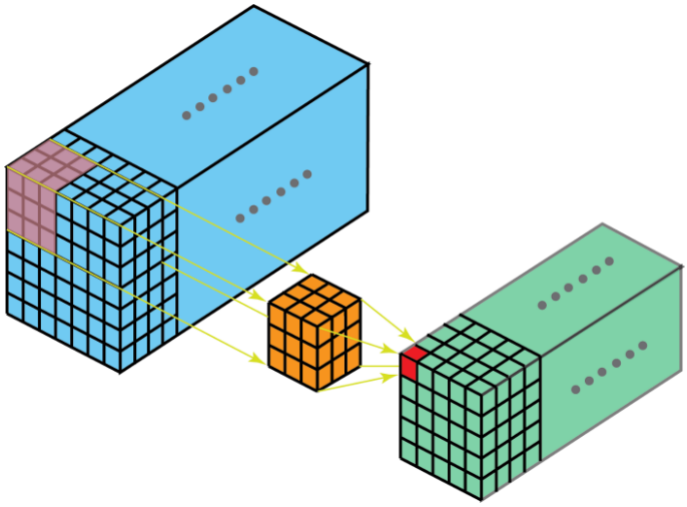


Figure 3.8: 3D convolutions describe the spatial relationships of objects in the 3D space [32]

are more suitable for spatio-temporal feature learning compared to 2D CNNs. The authors observed that C3D starts by focusing on appearance in the first few frames and tracks the salient motion in the subsequent frames. Figure 3.9 shows the deconvolution of two C3D conv5b feature maps with the highest activations projected back to the image space.[14]

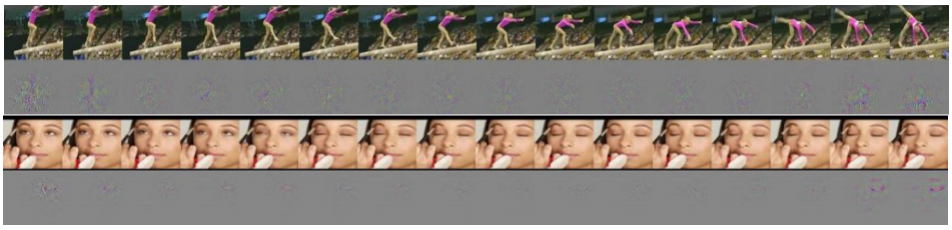


Figure 3.9: C3D captures appearance for the first few frames but thereafter only attends to salient motion [14]

3.2.5. 3D Residual Networks

3D ResNets [12] capitalized on the success of the 2D ResNet [10], and the ability of 3D CNNs to directly extract both spatial and temporal features. 3D ResNets were created by extending the 2D ResNet architecture and replacing the 2D convolutional filters with 3D kernels. 3D ResNets pretrained on Kinetics outperformed complex 2D networks on both UCF-101 and HMDB-51 datasets, as well as outperforming C3D.

4

Group Equivariant Convolutional Networks

Convolutional neural networks are equivariant to translations in the input data. This implies that any translations in the input data will also lead to translations in the output data. This plays a crucial role in deep learning as it helps reduce the number of trainable parameters while making the networks robust to translations in the input. The importance of learning equivariant representations in deep learning led to the advent of convolutional layers that utilise group symmetries to make CNNs equivariant to rotations and reflections. In this section, we shall discuss the concepts of equivariance and how it plays a pivotal role in the field of deep learning, as well as take a deeper look into the approach of group equivariance pioneered by Cohen and Welling [33], and how they are applied to convolutional networks.

4.1. Equivariance vs Invariance in CNNs

Equivariance allows networks to detect features within the input object inspite of transformations applied to it. To put it simply, a CNN does not require a feature to be at a fixed position in the input in order to be able to detect it. For a network to be **equivariant** to a certain symmetry function, it needs to satisfy the following conditions:

- The network should preserve the symmetries of the data - i.e. the outputs of the layers of the network should be able to retain the symmetries of the input.
- For any function that changes symmetry, the result obtained from applying the symmetry to the input first, followed by passing it through the network, should be the same as passing the input through the network first, followed by the symmetry function.

Mathematically speaking, if we consider the function that a network learns to be $f(x)$, the input to be x, and the symmetry function to be g, the above statements can be summarised by the equation:

$$f(g(x)) = g(f(x)) \tag{4.1}$$

Therefore, we can state that the function f is equivariant to the transformation represented by a symmetry function g, and if it holds for every symmetry function in a group of functions G, we can state that the function f is equivariant to the G.

Translational equivariance is achieved inside CNNs by the property of weight sharing. Since the same weights are shared across inputs, if a feature is present inside an input, it will be detected irrespective of its position This is extremely useful because it allows CNNs to perform classification and object detection tasks successfully. Figure 4.1 shows an example depicting the benefit of translational equivariance in CNNs, where multiple objects can be detected within the same input frame.

Translational invariance, on the other hand, makes the networks **invariant** to translations and is often confused with Translational equivariance. Translational invariance means that if the inputs undergo some translations, the network would still be able to identify the class to which the input belongs. In

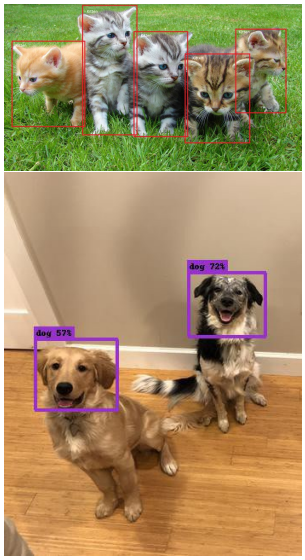


Figure 4.1: Property of equivariance in translation seen in object detection task. In the figures, the network is able to detect multiple instances of the animals inside a single frame

Convolutional neural networks, translational invariance is achieved through pooling operations. Pooling is the final step in a convolutional layer pipeline. The output of the network is replaced with an aggregate response such as the maximum or the average of the values in the region under the pooling kernel. Since the output is replaced with an aggregated response, the output from the pooling layer is not affected even if the input is translated.

Generalising, we can state that a network is **invariant** to a symmetry function, if,

- For any function that changes symmetry, the result obtained from applying the symmetry to the input first, followed by passing it through the network, or, passing the input through the network first, followed by the symmetry function is equal to the output of the network itself.

To represent this mathematically, let us again consider the function that a network learns to be $f(x)$, the input to be x, and the symmetry function to be g. The equation then becomes:

$$f(g(x)) = g(f(x)) = f(x) \tag{4.2}$$

The property of invariance proves to be extremely useful when the exact position of features within an input is not needed to determine which class the input belongs to. Figure 4.2 depicts an example of translational invariance in feature representations with the handwritten digit 2.

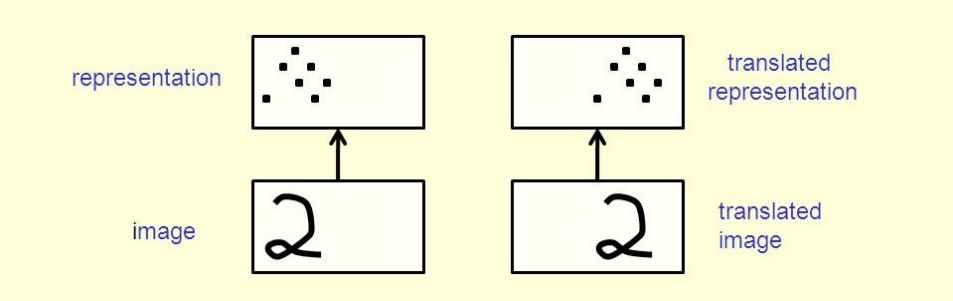


Figure 4.2: Translational invariance shown on the handwritten digit 2 along with its representation. Pooling the representation would not change the represented output, and the network would be able to classify the input correctly as the digit 2. [1]

So far, we have talked about equivariance in convolutional neural networks and provided a general idea of how it can be extended to more symmetry transformations. The following sections deal with how this idea evolves into the concept of Group Equivariant networks. To begin with, we shall briefly discuss the concepts of symmetry and group theory with respect to neural networks.

4.2. Symmetry

Symmetry in neural networks is perhaps best understood with respect to the task being performed by the network. In the case of image or action classification, we are concerned with predicting the correct class of the image or action. Therefore, a symmetry would be a transformation that preserves the label associated with the image or action. Let us consider that x is an input in a set of possible inputs I to a network, y is a function that maps the input x to a class label which serves as the ground truth for the network to learn. Therefore, based on our previous description, a function g is said to be in symmetry with the function y if,

$$y(g(x)) = y(x), \text{for all } x \epsilon I \tag{4.3}$$

The set of symmetries $g \epsilon G$ for which the above condition holds is called a group. However, before we dive into the details of how these groups relate to Group Equivariant convolutions, let us take a look at the general definitions of groups.

4.3. Groups

A group is defined as a set of elements and a binary operation which acts on the elements within that set. Let us consider a set G and a binary operation $*$. The group is, therefore, the set G on which the operation $*$ is performed and satisfies the following conditions:

- **Identity:** There exists an element $a \epsilon G$ such that, for every element $b \epsilon G$, $a * b = b * a = a$
- **Associativity:** For all a, b and $c \epsilon G$, $(a * b) * c = a * (b * c)$
- **Closure:** For all $a, b \epsilon G$, $a * b \epsilon G$
- **Inverse:** For each element $a \epsilon G$, there exists an element $b \epsilon G$ such that $a * b = b * a = e$

4.3.1. 2D Groups

Z2

Cohen and Welling, in their paper, considered a number of symmetry groups for two-dimensional spaces, i.e. for image classification tasks. The 2D groups are defined in terms of functions on the pixel spaces. Z2 represents the pixel space of an image, where a pixel in the space is identified by the coordinates $(u, v) \epsilon Z2$. We represent the group Z2 in the homogenous coordinate system as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.4}$$

It may be interesting to note that the above representations still consider only a single pixel in the respective space, but its usefulness lies in the fact that it allows the translation operation to be represented as matrices.

A translation group is defined as a group of all possible translations on the pixel space. A translation $d \epsilon D$ is a function which operates on Z2, and can be represented, in Z2 space as:

$$d(t) = t + (u, v) \tag{4.5}$$

where (u, v) represents the pixel coordinates in the Z2 space.

We can assume that in a pixel space, a translation operation performed on a pixel coordinate results in another pixel coordinate within that space. To put simply, if a translation d is applied to a pixel coordinate $(u, v) \epsilon Z2$, then the resultant pixel coordinates $(u_1, v_1) \epsilon Z2$.

Due to this association between the translation group D and the pixel space Z2, Cohen and Welling treat Z2 as D in two-dimensional space.

p4

The p4 group is composed of all **translations** and **rotations** by 90 degrees about any center of rotation in a two dimensional grid space. To conveniently parameterise the group, we consider three integer variables $r, u and v$, where r represents the number of times a grid is rotated by a factor of 90 degrees,

and (u, v) represent the translational component in Z2. We represent the group using these three parameters as follows:

$$g(r,u,v) = \begin{bmatrix} cos(\frac{r\pi}{2}) & -sin(\frac{r\pi}{2}) & u \\ sin(\frac{r\pi}{2}) & cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

where $0 \leq r < 4$, $(u, v)\ \epsilon\ Z2$.

The group operation is represented by a matrix multiplication and operates on the points in the two-dimensional pixel space $Z2$, by taking the product of the matrix $g(r, u, v)$ and the homogeneous coordinate vector $x(u_1, v_1)$ and is represented as:

$$gx = \begin{bmatrix} cos(\frac{r\pi}{2}) & -sin(\frac{r\pi}{2}) & u \\ sin(\frac{r\pi}{2}) & cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{4.7}$$

Figure 4.3 graphically represents the p4 group with all its rotations. The red arrows symbolise a clockwise rotation of 90 degrees, as seen in the image of the cactus.

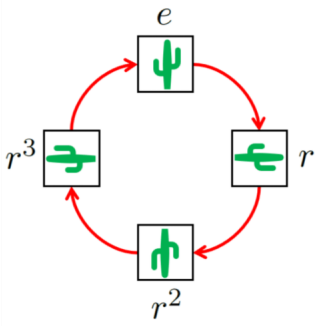


Figure 4.3: Graphical representation of the p4 rotations group depicting the identity object and its rotations by 90 degrees. Image sourced from [34]

p4m

The p4m group extends the p4 group by adding another parameter representing mirror reflections and is thereby composed of translations, rotations by 90 degrees and mirror reflections of the identity object. The p4m group is thus parameterized by four integer variables m, r, u and v, where m represents the mirror reflection, r represents the number of rotations by 90 degree, and $(u, v)\,\epsilon\, Z2$ represents the pixel coordinates. The group is represented by the four parameters as:

$$g(m,r,u,v) = \begin{bmatrix} (-1)^m cos(\frac{r\pi}{2}) & -(-1)^m sin(\frac{r\pi}{2}) & u \\ sin(\frac{r\pi}{2}) & cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

where $m\ \epsilon\ 0, 1$, $0 \leq r < 4$, $(u, v)\ \epsilon\ Z2$.

Figure 4.4 graphically represents the p4m group with its rotations and reflections. The red arrows symbolise a clockwise rotation of 90 degrees, and the blue lines represent the mirror reflections.

4.3.2. Functions on groups

Let us consider a symmetry transformation g, a symmetry group G, and a feature map f. In order to get a transformed version of f, it seems intuitive to write $g(f)$. However, it is important to note that the transformation g is defined to operate with pixels, and f is not a pixel. Therefore, a new function L_g is defined by Cohen and Welling to represent the same transformation, which can operate on feature maps. L_g is defined therefore defined mathematically as:

$$L_g f(x) = f \circ g^{-1}(x) = f(g^{-1}x) \tag{4.9}$$

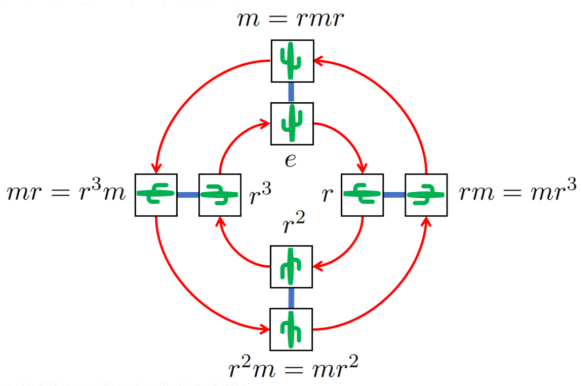


Figure 4.4: Graphical representation of the p4m group depicting the identity object, its rotations by 90 degrees and the mirror reflections. Image sourced from [34]

This means that in order to get the value of the transformed feature map $L_g f$ at the point defined by x, a lookup is done in the original feature map f, at the point $g^{-1}x$, providing the unique result which gets mapped to x by g. It may be interesting to note that the term g^{-1} has been used in the definition instead of g. A fairly intuitive explanation for this would be considering the transformation from a different perspective. If the transformation g represents the feature map being transformed over a field of pixels, then g^{-1} represents the pixels being transformed under the feature map. Thus, transforming a feature map by the symmetry operation g is the same as transforming the pixels by g^{-1}. For example, if g represents a pure translation, then g^{-1} represents a translation in the opposite direction.

The inverse of the transformation ensures that the function shifts in the positive direction if a positive translation is used, and allows the transformed feature maps to maintain homomorphism, i,e,

$$L_g L_h = L_{gh} \tag{4.10}$$

even for non-commutative transformations g and h, i.e. $gh \neq hg$.

4.4. Group Equivariant Networks

At every layer l, a convolutional network takes a stack of feature maps $f : Z2 \longrightarrow R^{K^l}$, as input and performs a convolution operation with a set of K^{l+1} filters $\psi^i ; Z2 \longrightarrow R^{K^l}$. We represent the convolution operation as:

$$\left[f \star \psi^i\right](x) = \sum_{y \,\epsilon\, Z2} f(x+y)\psi(y) \tag{4.11}$$

where both ψ and f have K channels.

4.4.1. Translational Equivariance with Groups

To recap, for the network to be equivariant, for any symmetry operation g, transforming the image f by g followed by convolution with ψ would give the same result as first performing the convolution of f with ψ, followed by the transformation g. In order to simplify, let us consider the number of channels $K = 1$, and the transformation g to be a pure translation. Thus, we aim to prove that:

$$[L_g f] \star \psi = L_g [f \star \psi] \tag{4.12}$$

Taking the left side of equation 4.12, we write $[L_g f] \star \psi$ as:

$$\begin{aligned} ((L_g f) \star \psi)(x) &= ((f \circ g^{-1}) \star \psi)(x) \\ &= \sum_{y \,\epsilon\, Z2} f(g^{-1}(x+y))\psi(y) \\ &= \sum_{y \,\epsilon\, Z2} f(x+y-g)\psi(y) \end{aligned} \tag{4.13}$$

We do the same for the right side of equation 4.12, as follows:

$$\begin{aligned}(L_g(f \star \psi))(x) &= (f \star \psi)(x - g) \\ &= \sum_{y \,\epsilon\, Z2} f((x - g) + y)\psi(y) \\ &= \sum_{y \,\epsilon\, Z2} f(x + y - g)\psi(y) \end{aligned} \tag{4.14}$$

From equations 4.13 and 4.14, we see that both the expressions are indeed equal, and is consistent with our definition of equivariance in equation 4.1.

4.4.2. Group Equivariant Convolutions

At this point, we have considered the equivariant convolutions for pure translations. The concept of equivariance, however, generalises with different symmetry groups such as the p4 and p4m groups. Let us revisit the mathematical representation of convolution defined in equation 4.11, and represent it as a convolution on a group:

$$(f \star \psi)(g) = \sum_{y \,\epsilon\, Z2} f(g(y))\psi(y) \tag{4.15}$$

where $g \,\epsilon\, G$ and the function $f \star \psi$ operates on the group G, instead of pixels. Therefore, equation 4.15 represents mathematically, the Group convolution for the first layer.

4.4.3. Deeper G-Convolutions

Previously, we defined the group convolution for the first layer of the network, but to build a deeper network of G-convolutions, we need to repeatedly convolve the result of the previous layers with more filters. Fortunately, that can be achieved by introducing a simple modification of our first G-convolution. Since f and ψ are defined on the group G after the first layer, the updated G-Convolution representation becomes:

$$(f \star \psi)(g) = \sum_{h \,\epsilon\, G} f(gh)\psi(h) \tag{4.16}$$

where $g, h \,\epsilon\, G$, and gh represents the composition $g * h$.

We can show that this definition of G-Convolution is also equivariant with respect to the group G. Let us consider a transformation $u \,\epsilon\, G$, such that

$$(L_u f) \star \psi = L_u(f \star \psi)$$

As previously done for the translations, we expand both sides of the equation as follows:

$$\begin{aligned}(L_u f) \star \psi(g) &= ((f \circ u^{-1}) \star \psi)(g) \\ &= \sum_{h \,\epsilon\, G} f(u^{-1}gh)\psi(h) \end{aligned} \tag{4.17}$$

$$\begin{aligned}L_u(f \star \psi) &= (f \star \psi)(u^{-1}g) \\ &= \sum_{h \,\epsilon\, G} f(u^{-1}gh)\psi(h) \end{aligned} \tag{4.18}$$

We see again from equations 4.17 and 4.18 that the two expanded forms are the same, and hence the deeper G-convolutions also preserve equivariance.

4.4.4. Implementation

The implementation of Group convolutions consists of indexing and inner products and can be broken down into two main steps - filter transformation and fast planar convolutions [35, 36, 37]. Cohen and Welling introduce the term *split* for the symmetry group if a transformation $g \,\epsilon\, G$ can be decomposed into translations $t \,\epsilon\, Z2$, and a transformation s which leaves the origin of transformation invariant. For instance, for the group p4, the transformation $g = ts$ can be decomposed into a translation t and a

rotation about the origin s, while p4m can be decomposed similarly into translations, rotations and mirror reflections. Using the relation defined in equation 4.10, we may update the representation of the G-convolution as:

$$f \star \psi(ts) = \sum_{h \,\epsilon\, X} \sum_{k} f_k(h) L_t[L_s \psi_k(h)] \tag{4.19}$$

where $k \epsilon K$ represents the filter, L_t is a translation performed on the image or feature map, L_s represents the transformation(here rotation for the p4 group) on the image or feature map, and $X = Z2$ for the first convolution layer, and $X = G$ for deeper layers.

To compute the p4 convolution $f \star \psi$, first the filter transformation $L_s \psi$ is computed for all four rotations, and then a fast planar convolution is computed on f and the filter bank.

Filter transformation
At a layer l, the array or tensor that stores the filters is of the shape

$$K^l \times K^{l-1} \times S^{l-1} \times *n \tag{4.20}$$

where K^l is the number of output channels of the layer l, K^{l-1} is the number of input channels to the current layer, S^{l-1} is the number of input transformations in G from the previous layer, known as the *input stabilizer size*, and $*n$ represents the filter kernel size. For the group p4, operating on 2D input, i.e. images, $*n$ is two dimensional, i.e. $n \times n$, $S^1 = 1$, and S^l where $l > 1$ is 4. Similarly for p4m, S^l is 8 for $l > 1$, while the other values remain the same.

Transformation of a filter with L_s leads to a permutation of the entries in the scalar filter channels in the filter bank F, producing an augmented filter F^+ with the shape:

$$K^l \times S^l \times K^{l-1} \times S^{l-1} \times *n \tag{4.21}$$

where S^l is the number of output filter transformations in G, and is also known as the *output stabilizer size*, and the rest of the terms have the same definition as before.

Planar convolutions
The next step in the Group convolution process is the planar convolution with the augmented filter F^+. For $S^{l-1} > 1$, i.e. for layers beyond the first convolutional layer, the sum over the space X defined in equation 4.19 sums over the stabilizer. Cohen and Welling state that this summation may be folded into a sum over the feature channels by reshaping the augmented filter bank F^+ from $K^l \times S^l \times K^{l-1} \times S^{l-1} \times *n$ to $K^l S^l \times K^{l-1} S^{l-1} \times *n$ thereby allowing it to be interpreted as a traditional convolution filter bank with $K^l S^l$ output channels and $K^{l-1} S^{l-1}$ input channels, and can be used to perform planar convolutions.

We have now discussed the pipeline of the group convolution, and it is nice to observe the group convolution process visually. Figure 4.5 shows the input and feature maps of a 2 layer group equivariant convolutional network with the transformation group p4. Both the identity input(the upright cactus) and the rotated input are provided to the network to visualise the represented feature maps better.

4.5. Three Dimensional Group Convolutions

Cohen and Winkels [38] developed Group Equivariant Convolutional networks for three-dimensional signals such as volumetric data, with the symmetry transformation groups consisting of translations, rotations and reflections in 3D space. The application of group equivariant convolutions in 3D space is interesting because the rotational and reflection groups in three dimensions are not commutative and lead to highly intricate structures.

To begin with, let us see how the volume space is represented in three dimensions. Just like in the case of two dimensional spaces, the 3D groups are defined in terms of functions on the voxel spaces [2]. The voxel space of volume data, such as CT scans with spatial volume or videos with spatio-temporal volume, is represented as **Z3** in the group terminology. As it is a point in three-dimensional space, a voxel is thus identified by the coordinates $(u, v, w) \,\epsilon\, Z3$. Z3 is represented in the homogeneous

[2] Short for volume pixel. It represents a single sample, or data point, on a three-dimensional grid.

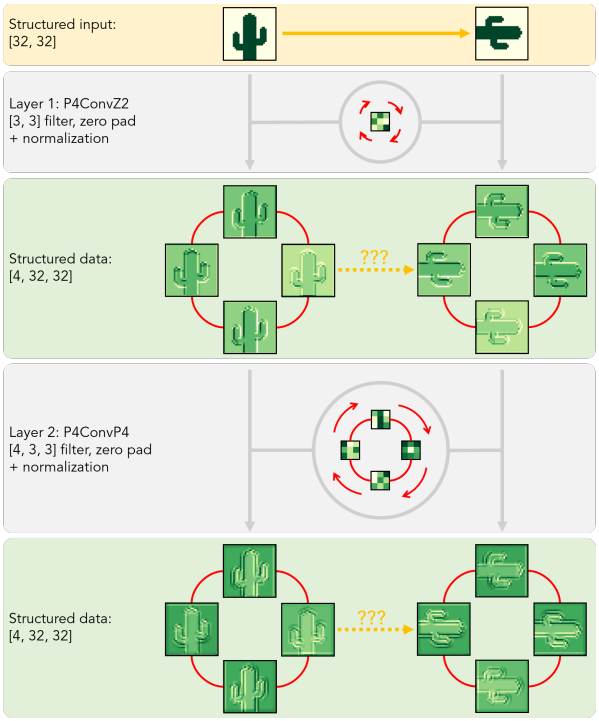


Figure 4.5: Representing the GConv pipeline and feature maps for a 2-layer CNN, using the image of a cactus in its identity form and rotated form. The generated feature map activations at each layer is shown for both layers. Image sourced from [34]

coordinate system as:

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \tag{4.22}$$

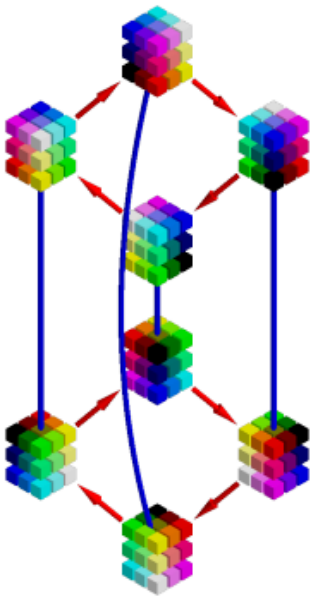
Intuitively, the translation group for Z3. is a group of functions which operate on the volume space, and can be represented, as:

$$d(t) = t + (u, v, w) \tag{4.23}$$

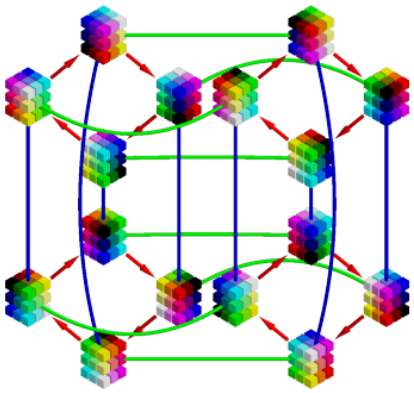
where (u, v, w) represents the pixel coordinates in the Z3 space. We make a similar assumption as in the case of the two dimensional space that a translation operation performed on a volume coordinate results in another volume coordinate within that space, i.e. if a translation d_3 is applied to a pixel coordinate $(u, v, w)\ \epsilon\ Z3$, then the resultant pixel coordinates $(u_1, v_1, w_1)\ \epsilon\ Z3$. Thus, we similarly consider the Z3 group to represent translations in three-dimensional space.

4.5.1. 3D Implementation

We have seen that in the simplest form of Group convolution, the group H is defined as a set of 2D planar rotations, and the group G includes H and translations in 2D space. Thus, four orientations per feature map undergo cyclic permutation on rotation. The complexity of 3D space lies in the fact that the filters are not squares but cubes or cuboids and lead to very intricate symmetry groups. Cohen and Winkels consider rotations and reflections in the 3D space and define four symmetry groups of interest - the orientation-preserving and non-orientation-preserving symmetries of a cuboid: D_4 and D_4h, and the orientation-preserving and non-orientation preserving symmetries of a cube: O and O_h. Figure 4.6 represents graphically the D_4 and D_{4h}, with each node representing a transformation on a $k \times k \times k$ filter. The figures consist of arrows and lines connecting the nodes representing the rotations and mirror reflections on the different axes.



(a) Graphical representation of the D_4 group.



(b) Graphical representation of the D_{4h} group.

Figure 4.6: Graphical representations of the groups D_4 and D_{4h} using Cayley diagrams. Each node is a transformation of the group $h \epsilon H$, and is represented by the orientations of a $3X3X3$ filter. Images sourced from [**?**]

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] Y. Upadhay, “Introduction to feedforward neural networks.” [Online]. Available: https://towardsdatascience.com/feed-forward-neural-networks-c503faa46620

[4] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *Under Review of ICLR2016 (1997)*, 11 2015.

[6] A. L. Maas, “Rectifier nonlinearities improve neural network acoustic models,” 2013.

[7] “Learning rate decay and methods in deep learning.” [Online]. Available: https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f910b

[8] “Learning rate schedules and adaptive learning rate methods for deep learning.” [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

[9] “Introduction to feedforward neural networks.” [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2015.

[12] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri, “Convnet architecture search for spatiotemporal feature learning,” *arXiv preprint arXiv:1708.05038*, 2017.

[13] J. Carreira and A. Zisserman, “Quo vadis, action recognition? a new model and the kinetics dataset,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[14] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[15] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016.

[16] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” 2018.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Advances in Neural Information Processing Systems*, vol. 3, 06 2014.

[19] B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370281900242

[20] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, “Hmdb51: A large video database for human motion recognition,” 11 2011, pp. 2556–2563.

[21] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.

[22] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, T. Yan, L. Brown, Q. Fan, D. Gutfruend, C. Vondrick, and A. Oliva, “Moments in time dataset: one million videos for event understanding,” 2019.

[23] W. Price and D. Damen, “Retro-Actions: Learning ’Close’ by Time-Reversing ’Open’ Videos,” p. 10.

[24] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, “Long-term recurrent convolutional networks for visual recognition and description,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[25] Z. Qiu, T. Yao, and T. Mei, “Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks,” *arXiv:1711.10305 [cs]*, Nov. 2017, arXiv: 1711.10305. [Online]. Available: http://arxiv.org/abs/1711.10305

[26] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, “Action recognition in video sequences using deep bi-directional lstm with cnn features,” *IEEE Access*, vol. 6, pp. 1155–1166, 2017.

[27] B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, no. 1, pp. 185 – 203, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370281900242

[28] S. Beauchemin and J. Barron, “The computation of optical flow.” *ACM Computing Surveys (CSUR)*, vol. 27, pp. 433–466, 09 1995.

[29] U. of Washington CSE Vision Group, “Lecture 15 - motion,” https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect15.pdf.

[30] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in neural information processing systems*, 2014, pp. 568–576.

[31] Shuiwang Ji and Wei Xu and Ming Yang and Kai Yu, “3d convolutional neural networks for human action recognition.” [Online]. Available: https://www.dbs.ifi.lmu.de/~yu_k/icml2010_3dcnn.pdf

[32] Kunlun Bai, “A comprehensive introduction to different types of convolutions in deep learning.” [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep\-learning-669281e58215

[33] T. S. Cohen and M. Welling, “Group Equivariant Convolutional Networks,” *arXiv:1602.07576 [cs, stat]*, Jun. 2016, arXiv: 1602.07576. [Online]. Available: http://arxiv.org/abs/1602.07576

[34] “Geometric deep learning: Group equivariant convolutional networks.” [Online]. Available: https://medium.com/swlh/geometric-deep-learning-group-equivariant-convolutional-networks-ec687c7a7b41

[35] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, “Fast convolutional nets with fbfft: A gpu performance evaluation,” 2015.

[36] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through ffts,” 2014.

[37] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” 2015.

[38] M. Winkels and T. S. Cohen, “3D Group-Equivariant Neural Networks for Octahedral and Square Prism Symmetry Groups,” p. 5.

[39] “The unreasonable effectiveness of recurrent neural networks.” [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[40] “Understanding lstm networks.” [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[41] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.

List of Figures

2.1 A multi-layered perceptron with a depth of 3, represented as a directed acyclic graph. The learnable parameters are represented by the vector W for the mapping between x and h, and the vector w for the mapping between h and y. Image taken from [**?**] 13
2.2 A multi-layered perceptron with a depth of 5, consisting of 3 hidden layers. Image taken from [3] 14
2.3 Plots of Rectified Linear Unit(ReLU) and Leaky ReLU. The plots are obtained using an input $x \epsilon [-10, 10]$ and a coefficient of 0.05 for the LeakyReLU activation function. 15
2.4 Gradient descent algorithm showing the global minimum of the function $f(x) = \frac{1}{2}x^2$ [**?**] 16
2.5 Comparing optimizations with constant learning rate and decayed learning rate. Figure sourced from [7] 17
2.6 Learning rate reduction using step decay. Image sourced from [8] 17
2.7 Convolutional neural network to classify handwritten digits. [9] 18
2.8 Simple example of a 2D Convolution performed on an image of size 4x3 using a kernel of size 2x2 with a stride of 1. [2] 19
2.9 Convolution kernels being applied to images and the resultant activation maps highlighting the detected features. [**?**] 19
2.10 Max pooling on two different images of the handwritten digit 5, showing activated detector units. [2] 20
2.11 An overview of a convolution layer containing a convolution block with three filters, a ReLU non-linear activation block and a spatial pooling block. [**?**] 20

3.1 Examples from popular video action datasets. At the top we have single frames from different classes from the UCF-101 dataset, followed by single frames from classes in the HMDB-51 dataset, and the Kinetics dataset respectively, and at the bottom we have consequtive frames from the Something-Something dataset for two classes. 23
3.2 Video representation using CNN and LSTMs 24
3.3 Predictions of LSTM video representation method for action recognition for sample clips. Wrong classifications are marked in red.[26] 25
3.4 Depicting optical flow field of the image of the Rubik's Cube between two temporal instances [29] 25
3.5 Two-stream architecture for video classification [30] 26
3.6 Optical flow of actions performed in a video in temporal domain [30] 26
3.7 Methods of fusing the parallel streams in Two-Stream Networks. Image sourced from [13] 26
3.8 3D convolutions describe the spatial relationships of objects in the 3D space [32] 27
3.9 C3D captures appearance for the first few frames but thereafter only attends to salient motion [14] 27

4.1 Property of equivariance in translation seen in object detection task. In the figures, the network is able to detect multiple instances of the animals inside a single frame 29
29figure.caption.29
4.3 Graphical representation of the p4 rotations group depicting the identity object and its rotations by 90 degrees. Image sourced from [34] 31
4.4 Graphical representation of the p4m group depicting the identity object, its rotations by 90 degrees and the mirror reflections. Image sourced from [34] 32
4.5 Representing the GConv pipeline and feature maps for a 2-layer CNN, using the image of a cactus in its identity form and rotated form. The generated feature map activations at each layer is shown for both layers. Image sourced from [34] 35

4.6 Graphical representations of the groups D_4 and D_{4h} using Cayley diagrams. Each node is a transformation of the group $h \epsilon H$, and is represented by the orientations of a $3X3X3$ filter. Images sourced from [**?**] 36

A.1 Inside a Recurrent Neural Network 44
A.2 Types of RNNs - Each rectangle is a vector and arrows represent functions. Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state.[39] 44
A.3 Inside a LSTM network [40] 45

List of Tables

B.1 Distribution of the workload 46

A

RNNs and LSTMs

A.1. A Brief Recap about Recurrent Neural Networks

The limitation of Vanilla neural networks is that it has a fixed-sized input and a fixed-sized output. Recurrent Neural Networks or RNNs are a particular type of neural networks that allow you to model *sequences* of vectors that appear in the input or output of an image. RNNs maintain a given state over time as described below.
For a sequence $\boldsymbol{X_t}$, for t = 1, 2, 3, ..., T, the activation at the current step is given by,

$$\boldsymbol{H_t} = \phi(\boldsymbol{X_t W_{hx}} + \boldsymbol{H_{t-1} W_h h} + \boldsymbol{b_h}) \tag{A.1}$$

and the output is given by

$$\boldsymbol{O_t} = \boldsymbol{H_t W_{hq}} + \boldsymbol{b_q}\,. \tag{A.2}$$

We define the components of equations A.1 and A.2 as follows:

- $\boldsymbol{H_t}$: activation at current time step *t*,
- ϕ: non-linearity,
- $\boldsymbol{X_t}$: input at time *t*,
- $\boldsymbol{H_{t-1}}$: activation at time step *t-1*,
- $\boldsymbol{W_{hx}}$, $\boldsymbol{W_{hh}}$, $\boldsymbol{W_{hq}}$: learned weights,
- $\boldsymbol{b_h}$, $\boldsymbol{b_q}$: learned bias terms,
- $\boldsymbol{O_t}$: output.

The same parameters are shared at each time step. This process is visualized in figure A.1 for better understanding.

There are five types of RNNs as can be seen in figure A.2. The one-to-one network is the standard, or ”vanilla” neural network which takes in a fixed-sized input and produces a fixed-sized output, and is used in image classification. The one-to-many network takes a fixed-sized input and gives a sequential output, and an application of this network is for image captioning, where the input is an image and the output is a sequence of words. The many-to-one network takes a sequential input and gives a fixed-sized output, applied in classifying an action where the input is multiple video frames and the output is an action. There are two types of many-to-many networks. The first one is a sequential input and sequential output network, which can, for instance, be used in translation where an RNN is fed a sentence in English and it outputs the sentence in Spanish. The second type of many-to-many network is when the input and output are synced and sequential, which can be used in video classification to label each frame of the video.

A.2. A Brief Recap about Long Short-Term Memory Networks

One of the unique advantages of RNNs over vanilla neural networks is its ability to hold state information. As a result of this, we are able to gain context from previous tasks to the current tasks. For

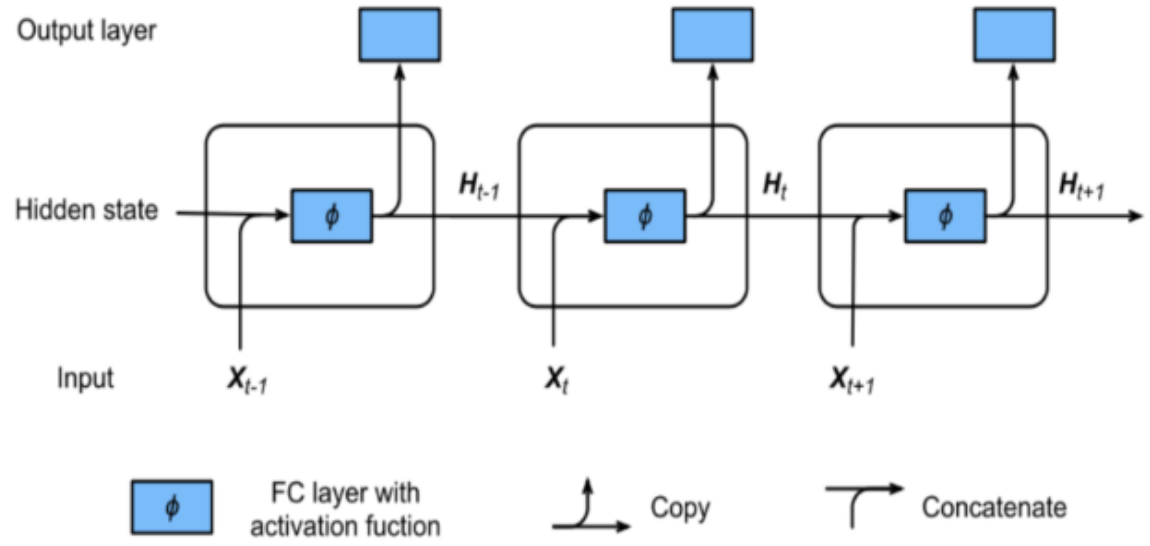


Figure A.1: Inside a Recurrent Neural Network

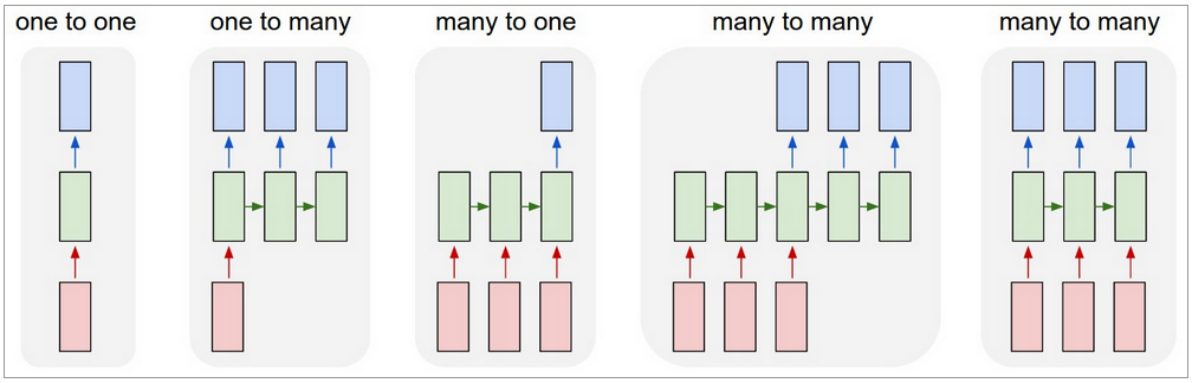


Figure A.2: Types of RNNs - Each rectangle is a vector and arrows represent functions. Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state.[39]

instance, a RNN can classify an action based on a series of video frames, as it holds context over all the given video frames to predict an action. But RNNs cannot hold memory for long. In the context of a text completion task, if the RNN is given a short sentence like ”The sky is ...” and was asked to complete it, it would return ”blue”. Instead, if it was given a long sentence like ”I grew up in Spain, and moved to the Netherlands a week ago. I speak fluent ...” and was asked to complete it, the RNN would return ”Dutch” in this context as the most recent contributing information was ”the Netherlands”. To solve these kinds of tasks, RNNs need to store memory over a long term to provide more context to tasks. This is where Long Short-Term Memory networks[41], or LSTMs, come into play.

LSTMs are a variant of RNNs that are capable of learning long-term dependencies. The core of the LSTM is the cell state, called Memory in figure A.3. The information flows directly from timestamp *t-1* to *t*, with minor linear modifications. The LSTM has the ability to control the information passing through the cell state with the help of *gates*. A LSTM has three gates: a forget gate, an input gate, and an output gate. Let us look at these three gates in detail.

The forget gate - The forget gate determines what information we will be removing from the cell state. This is done with a sigmoid layer which outputs a number between 0 and 1. A 0 represents that all the information in the cell state is thrown away, while a 1 represents that the information in the cell state

remains intact.

The input gate - Next, we determine what new information we are going to add to the cell state. The ”input gate layer” decides what values are to be updated, while the **tanh** layer creates a vector of new candidate values, $\tilde{C}_t$ that could be added to the cell state.

The output gate - The output gate outputs a filtered version of the cell state. The cell state is run through a sigmoid layer that decides what parts of information from the cell state we are going to output. Then we put the cell state through a **tanh** layer and multiply it by the output of the sigmoid gate.

The advantage of cell state is that during backpropagation there is a better flow of gradients as it does not go through all the gates inside the cell. This also solves the gradient vanishing problem common in RNNs. The gradient vanishing problems is when the gradients are small in magnitude, and when they are carried over a long sequence, due to the multiple multiplications, the gradients will approximate to 0, causing them to ”vanish”. The vanishing gradients problem is a hurdle in training RNNs, but in LSTMs, due to the direct flow of cell state from one cell to the next, the vanishing gradients problem is solved.

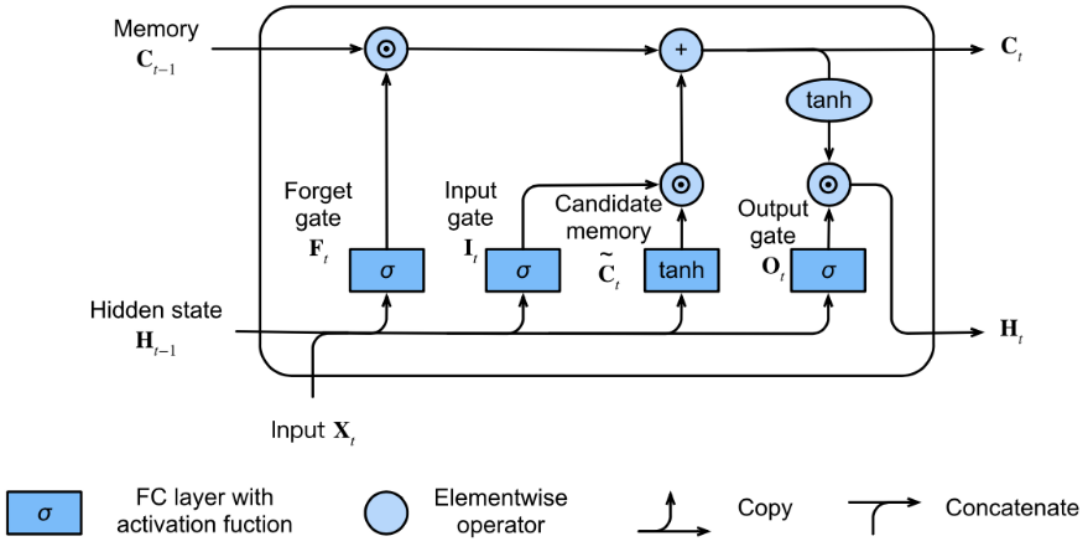


Figure A.3: Inside a LSTM network [40]

B

Task Division

Table B.1: Distribution of the workload

	Task	Student Name(s)
	Summary	
Chapter 1	Introduction	
Chapter 2		
Chapter 3		
Chapter *		
Chapter *	Conclusion	
	Editors	
	CAD and Figures	
	Document Design and Layout	