Learning Phase-Based Descriptions for Action Recognition

by

Omar Hommos

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday May 31, 2018 at 15:30 PM.

Student number: 4627857
Thesis committee:
Prof. dr. ir. M.J.T. Reinders , TU Delft, chair
Dr. ir. J.C. van Gemert, TU Delft, supervisor,
Dr. ir. S.L. Pintea, TU Delf, daily supervisor,
Dr. ir. J. Urbano, TU Delft

TU Delft

Preface

This report documents the findings of my MSc thesis report. The main content of the report is the scientific paper. It includes the motivation, methodology, implementation, and results of this research project. The appendix includes a brief review of the scientific background needed to understand this report.

I would like to thank my supervisors, Dr. Jan van Gemert and Dr. Silvia-Laura Pintea. Their support was not only continued, but also very immediate. Their supervision helped me sharpen my critical research skills, and allowed me to develop and test my ideas independently. Many thanks go to my best friends Ali F., Ali Z., Abdulhadi, Sharief, Yahya, Fituri, and my study colleague Usman.

Most importantly, none of this would have been possible without the support of my family. My parents and sisters have been there for me since day one, and continue to be by my side no matter what.

Omar Hommos
Delft, May 2018

Contents

1

Scientific Paper

Learning Phase-Based Descriptions for Action Recognition

Omar Hommos
Computer Vision Lab, TU Delft
o.hommos@student.tudelft.nl

Abstract

Action recognition continues to receive significant attention from the research community, with new neural network architectures being developed continuously. Optical flow is by far the most popular input motion representation to these architectures, leaving a lot of undiscovered potential for other types of motion representations. Eulerian representations have recently showed huge improvements in areas like motion magnification and video frame interpolation. This work proposes using a phase-based approach to make the best out of Eulerian motion information. We do this by learning complex filters using complex convolutional layers. Phase descriptions are extracted from the feature maps of these complex layers, and are then passed to the remainder of the convolutional network. Our approach shows great potential, and its performance exceeds that of a single optical flow frame input. We provide detailed analysis on using phase-based methods for Eulerian representations, in addition to further analysis on using Eulerian phase, rather than Lagrangian optical flow, for action recognition.

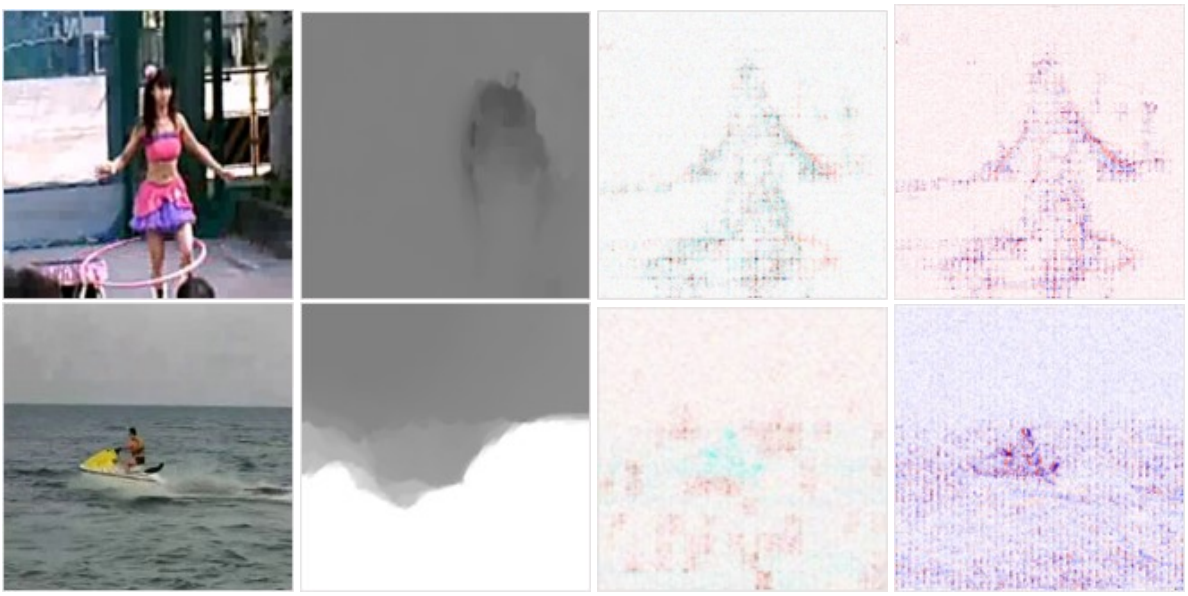
1. Introduction

Action recognition in videos is a challenging task that continues to receive a significant amount of attention from the research community. As with other areas in computer vision, most recent advances in action classification rely on training Convolutional Neural Networks (ConvNets) on video datasets [3, 20, 28, 32]. Contrary to still images, which contain only spatial information, temporal information from video frames provide additional and important clues for classification. Many actions can be identified using only spatial information *e.g*. detecting a Piano in a video strongly suggests that the action is *Playing Piano*. However, temporal information is often very important *e.g*. both *Basketball Throw* and *Basketball Dunk* actions contain basketballs, thus motion patterns are the determinant for recognizing the action.

To help ConvNets focus on learning and extracting temporal information, motion representations are used as inputs [28, 26]. The most popular motion input is optical flow [28, 3, 35, 36, 26], where the trajectory of a pixel between two consecutive frames is calculated *i.e*. a pixel-value is tracked over time. Other less popular representations include difference of RGB frames [35], and motion vectors [38]. Another option is to simply stack a sequence of images. This is not a motion representation per se, but does contain latent motion information. The network then takes responsibility for manifesting this implicit information for classification purposes. Fig. 1 shows examples of several input types. A considerable body of work studied and improved networks that use optical flow and stacked images for action recognition [3, 37, 26, 8, 22]. However, the use of other types of inputs, like Eulerian representations, is not well studied yet. This work aims to utilize and analyze Eulerian representations in the context of action recognition.

Eulerian representations. There are two main specifications for a motion field: *Lagrangian*, where a pixel value is tracked through time, and *Eulerian*, which focuses on value changes at a specific pixel location through time. Fig. 2 illustrates the difference. Optical flow belongs to the family of Lagrangian representations, while difference of RGB or difference of grayscale frames are Eulerian. Eulerian representations compare favorably to Lagrangian for action recognition:

- Lagrangian (optical flow) requires point matching; It fails on mismatches (*e.g*. if a person rotates), and untextured surfaces (*e.g*. water, hula hoops). In contrast, Eulerian representations focus on change per pixel-location, and are not susceptible to this.
- Due to their nature, directional and multi-scale information are easier to extract from Eulerian representations. However, interpretation of trajectory information at a pixel location requires looking at the complete neighborhood. This neighborhood becomes relatively large for high-velocity actions, which could pose a challenge to some classification approaches.
- Sevilla-Lara *et al*. [26] showed that the accuracy of



(a) RGB (b) Optical Flow (c) dRGB (d) dPhase

Figure 1: **Example images of different possible inputs** (a) RGB. (b) Optical flow. (c) Difference of RGB (dRGB), inverted. (d) Difference of phase (dPhase), will be further explained in section 5; red indicates a positive value, while blue indicates a negative value.

an optical flow method at boundaries and at small displacements is most correlated with action recognition performance. They also showed that optical flow fields which minimizes action recognition error focus on motion inside the body and at its boundaries. Since Eulerian representations measure changes per pixel location, they inherently model these details accurately.

In previous work, RGB difference was considered [35], but only as an input to the same network used for images or optical flow. Performance of an input is very dependent on the classification network [3]. Hence, architecture-level changes could be necessary to make full use of Eulerian motion information. Phase-based methods proved to be effective in many areas in computer vision. This includes motion estimation [9, 11], where image velocity is calculated from the phase of the response of a sequence of images to a fixed set of complex quadrature filters [9]. We adapt this idea to ConvNets. Specifically, this work proposes a phase-based network to make a better use of Eulerian inputs. This is done by learning complex filters in the first N complex convolutional layers of the network, then taking the phase of the last complex activations to obtain motion descriptions. The remaining layers take care of classifying extracted description. The whole network is trained end-to-end.

The contributions of this work are as follows: 1) Using phase of complex activations from learned filters for action description; 2) Extracting phase contours using perpendicular orientation filters; 3) An empirical analysis of the advantages and failure cases for the phase-based approach; 4) An empirical analysis of using Eulerian phase, rather than Lagrangian optical flow, for action recognition.

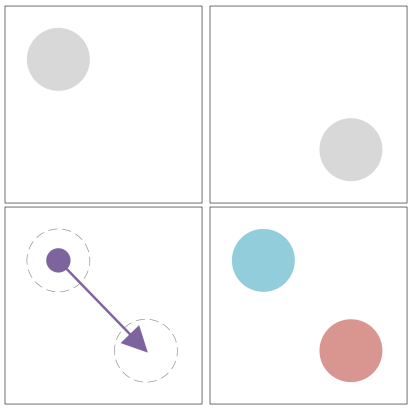


Figure 2: **Eulerian vs Lagrangian.** *Top:* frames 1 and 2, illustrating a pixel point moving diagonally. *Bottom left:* Lagrangian description, tracks the point (pixel value), gives a displacement vector. *Bottom right:* Eulerian description, measures change at each pixel location. Red and blue indicate positive and negative change of value respectively.

2. Related Work

Learning action recognition. To capture temporal structure, top performing architectures use two-stream networks, 3-Dimensional (3D) ConvNets, or Recurrent Neural Networks (RNNs). One of the early best performers is the two-stream network [28], where stream-one (*the spatial stream*) focuses on learning spatial information as its input is a single frame, while stream-two (*the temporal stream*) focuses on learning temporal information, and its input is a motion representation i.e. optical flow. Streams can each be trained separately, and softmax scores are then averaged to get a prediction. They can also be trained jointly, by computing a joint loss with the averaged softmax scores, and using it for training. A stream can adapt popular architectures, like VGG-16 or ResNets, with the possibility of adding inter-stream fusion, to create correspondences between feature maps of both streams [8, 7]. In this work, we adapt a two-stream architecture for out final network implementation.

A natural extension to 2D ConvNets for image classification is 3D ConvNets for video classification. 3D convolutions can learn to extract rich spatiotemporal features from videos. [32] started with a custom architecture of 3D ConvNets, and delivered decent performance. However, working with 3D ConvNets is often challenging, due to the large amounts of data needed to learn meaningful features without overfitting. In [3], authors started with the Inception architecture [30] that proved to deliver high performance with images, and inflated each 2D operation to a 3D operation, while using scaled pre-trained ImageNet kernels to initialize 3D kernels. This architecture, named *I3D*, coupled with a new relatively-large dataset named Kinetics [20], showed the power of 3D ConvNets for videos, and delivered state-of-the-art performance. Work in [37] proposed replacing 3D convolutions that use $k_t \times k \times k$ filters with $1 \times k \times k$ followed by $k_t \times 1 \times 1$ filters, to significantly increase computational efficiency. They also proposed adding feature gating, to capture dependencies between feature channels with a simple multiplicative transformation, which further increased performance. In this work, we discuss the extension of our approach to 3D ConvNets, and the importance of this extension to performance.

Lastly, RNN-based architectures can be used to aggregate features extracted from input videos, and model actions with large temporal scale. Those features can be obtained by training a ConvNet on a sequence of images or optical flow fields [5, 23]. Our proposed architecture can be used as the feature extraction block in such systems. Feature aggregation is often done using Long Short-Term Memory (LSTM) networks, a special kind of RNNs [5, 23].

Motion representations for action recognition. Previous work that experimented with different inputs focused on replacing optical flow to foster real-time recognition [35, 38], or learning to generate a Lagrangian representation specific for action recognition [22, 26].

To increase inference speed, the work in [35] experimented with using RGB difference, as calculating flow is quite time consuming [35, 38]. It did not outperform optical flow, yet still obtained considerably good results at a significantly higher processing speed ($\times 25$ faster inference). The use of motion vectors was proposed in [38]. Motion vectors are Lagrangian representations that are similar to optical flow, but capture only coarse-motion, and are typically used for video compression. It did not outperform optical flow, but still obtained comparable results at $\times 27$ faster processing time than two-stream networks [28].

Instead of explicitly using a motion representation as an input, it is possible to cascade a network that learns to generate optical flow [26, 22] or similar motion representations [39] before the classification network. In [26], an action recognition network is cascaded after an optical flow estimation network. The system is trained end-to-end. Thus, the flow network learns to generate flow fields targeted towards action recognition. Generated fields using this method minimize classification error, not the end-point-error (the performance measure for nominal optical flow estimation methods). A similar approach was used in [22], where the classification accuracy was used as a loss to fine-tune the optical flow network. The work focused on learning motion from unlabeled videos, with a significant improvement in action recognition accuracy over previous unsupervised methods. Finally, the work in [39] also added a cascaded motion estimation network to their model. However, it did not focus explicitly on forcing this estimation network to generate optical flow. The estimated representation was visually similar to optical flow in general, except at dynamic textures like water, which are challenging for flow estimation in general. Trying to force this input to be similar to optical flow by adding extra information from pre-calculated flow fields caused a drop in performance. Explicit use of optical flow as an input remains superior over other methods for action recognition. However, and knowing the downsides of using optical flow, the search for the best motion representation for action recognition remains an open question.

Phase-based methods. The idea of using phase information appears recurrently in computer vision, as it is very robust against blur and lighting changes. This has motivated its use to measure image velocity and reconstruct optical flow fields. Optical flow reconstruction is done by extracting phase changes from image sequences and solving for velocity, either deterministically using a set of equations [9], or using an RNN-based system [11]. Measuring phase difference of a stereo image pair helps in estimating disparity at each pixel location [10]; an important step for depth

estimation. In [34], motion magnification of small movements was possible by measuring phase variations of the complex steerable pyramids of a sequence of images, and then magnifying motion in a reconstructed video by modifying the measured phase changes.

Phase-based ConvNet architectures appeared very recently in literature. In [24], a ConvNet architecture was proposed to encode, manipulate, and then decode two subsequent input frames, to obtain an output frame with magnified motion. The encoder acts as a spatial decomposition filter and learns to extract a motion representation analogous to the phase of complex steerable pyramids of [34]. PhaseNet was proposed in [21], which is a decoder network that receives the decomposition of two input frames (the result of applying steerable pyramid filters), and tries to predict the decomposition of the predicted frame. A phase loss function is used for training, and results compare favorably to classical phase-based interpolation algorithms, and other learning-based interpolation networks. Just like recent learning-based work with phase, we adapt ideas that are regularly used in classical phase-based motion measurement and modeling approaches.

2.1. Revisiting Related Work: Effective learning from motion representations

The effectiveness of learning motion information depends on the used representation, the quality and size of the data, and the architecture.

Representation matters. Optical flow proved its effectiveness, as several state-of-the-art networks rely on it for actions recognition [3, 28, 8, 37]. Using a stack of several optical flow images gives a non-trivial increase in performance over using one image. In contrast, stacking images only trivially increases the performance of 2D ConvNets [19, 28], suggesting that networks were not making a good use of the hidden motion information. This illustrates the importance of having a good data representation.

Data matters. One of the biggest problems in learning-based approaches to action recognition is the lack of sufficient training data. This causes networks to quickly overfit, hence the importance of using many forms of data augmentation, aggressive regularization (*e.g.* a dropout ratio of 0.9), and ImageNet pretraining for most networks. Lack of data is especially problematic when raw frames are used as inputs (be it a single frame for spatial streams, or a stack of frames for temporal streams).

The invariability of optical flow to appearance, as seen in Fig. 1, contributes much more to performance than its trajectory information [26]. This helps it suffer less from overfitting, and perform generally better than raw frames. However, with the introduction of more and diverse data, like the Kinetics dataset [20], raw frames performance exceeds that of the optical flow [3, 37]. Hence, the performance of a representation relies heavily on the amount and diversity of available data.

Architecture matters. Given sufficient data, 3D ConvNets on stacked frames learn to extract rich spatiotemporal features, and outperform stacked flow fields (71.1% vs 63.4% on Kinetics, for the I3D architecture[3]). But how did the choice of architecture affect learning temporal information? Taking I3D (a 3D ConvNet architecture) and transforming all its 3D convolutions to 2D (effectively having 2D convolutional layers, stacked point-wise temporally) gives an architecture, called I2D [37], whose performance shows the importance of 3D convolutions. I2D continues to have 3D pooling between deep layers, so it is not a pure 2D model. Performance for I3D vs I2D is 71.6% vs 67%. When frames are shuffled during testing, thus temporal information is removed, I3D performance drops to to 45.37%, whereas I2D performance remains the same. This indicates that it learns no temporal features, and shows the importance of properly designing architectures to make full use of the given data.

Tests on combining parts of the I2D and I3D architectures show that while extracting low-level local temporal features from images in early layers is important, models that extract temporal information from high-level features in deeper layers are more accurate [37]. Furthermore, visualization of 2D and 3D filters for the temporal streams (operating on optical flow) of the two-stream [28] and the I3D models [3] shows significantly better use of local spatiotemporal information by 3D ConvNets. All this should be considered when designing motion-based architectures for action recognition, and when comparing representations.

Significance of using motion. Spatial information can always be inferred from the structure of motion fields [18] *e.g.* flow vectors that describe hand movement do imply the shape of the hand. Thus, it is expected that the temporal stream of a two-stream network will also learn spatial information. However, the removal of temporal information by shuffling test frames shows that spatial data is the largest contributor to performance, and not motion data [37, 26].

Most related work did not carry an extensive analysis on how and when does motion contribute to correcting an incorrect prediction, or vice versa. In addition, not all architecture-related motivations are revisited. For example, the work in [8] motivates fusing information from several neighboring temporal samples, to increase the performance of long-term actions, and learn context over a larger temporal scale [8] (*e.g. Archery* action: drawing an arrow, bending a bow, then shooting an arrow). The analysis did not include revisiting the performance of those specific action classes.

Performance increase could be a result of simply using more samples, rather than effectively learning long-term context of actions. Lack of such analysis can be attributed to not having standardized specifications for dataset classes *e.g.* when and which actions classes are considered long term? In this work, we provide a thorough analysis on the impact of using a phase-based approach, and the impact of using Eulerian descriptions. We overcome the lack of specifications for dataset classes, which is very important for correct analysis, by incorporating the analysis of the well understood phase-based methods that we base our work on.

3. Phase-Based Motion Description

Fourier's shift theorem states that a shift in time domain corresponds to a related linear phase shift in the frequency domain. As an example, the translation of an object through a sequence of frames, causes a corresponding variation in the phase of the Fourier domain of these frames. Since phase variations directly correspond to motion, they could serve as a viable representation of it. This does not only apply to the phase of the Fourier basis functions (sine waves), but to the phase of other complex representations as well, like complex-steerable pyramids [34].

3.1. Component Velocity from Phase Contours

Properties discussed above can be exploited to estimate the 2D velocity $V(v_1, v_2)$ at position $X(x_1, x_2)$ in the image at time t, *i.e.* estimate the optical flow field [9, 11].

Initially, images are represented by the response to a set of linear shift-invariant filters. Each filter is tuned to a narrow range of orientation, speed, and scale, and has only local spatiotemporal support. The response of a filter over a certain spatiotemporal patch relates to the existence of motion in that exact image patch, with the filter's respective orientation, speed, and scale tuning. Ideally, these velocity-tuned filters span the frequency domain to provide a complete representation of the image. From this initial representation, component velocity from each filter response is then calculated, and the flow field can be reconstructed.

Let $F(x,t)$ be a complex-valued spatiotemporal quadrature filter, that exhibits constant phase properties. Its response, $R(x,t)$, can be expressed using its magnitude and phase components, denoted as $\rho(x,t)$ and $\phi(x,t)$ respectively;

$$R(x,t) = \rho(x,t)e^{i\phi(x,t)} \tag{1}$$

$$\rho(x,t) = \big|R(x,t)\big| = \sqrt{Re[R(x,t)]^2 + Im[R(x,t)]^2} \tag{2}$$

$$\phi(x,t) = arg[R(x,t)] \tag{3}$$

Fleet and Jepson [9] showed that the temporal evolution of contours of constant phase provide a good approximation to the motion field. To elaborate, there exist points X that lie on a constant phase contour that evolves through time. These points satisfy $\phi(X,t) = c$, where c is a constant. Differentiation with respect to time shows a relation between phase gradients and velocity:

$$\begin{aligned}\Delta\phi(X,t) \cdot \Delta X &= \Delta\phi(X,t) \cdot (V,1) \\ &= (\phi_X, \phi_t) \cdot (V,1) = 0\end{aligned} \tag{4}$$

Where $\Delta\phi(X,t)$ is the temporal phase gradient vector, and V is the 2D velocity to be estimated [9]. To simplify the process, spatiotemporal filters can be replaced by spatial filters on each image to extract spatial gradients $\phi(X)$. Temporal phase gradient information are then extracted from $(\phi(X), t)$ pairs using an optimization technique [11].

After obtaining temporal phase gradients, further work is needed to fully recover velocity estimations. However, in the context of action recognition, there is no need to exactly compute velocity values, but rather *describe* the motion using the evolution of phase contours for classification purposes. Hence, extracted phase gradients can be used immediately as inputs to classifiers.

An adapted version of the explained process [9, 11] is the base to our phase-based approach for action recognition. Namely, we propose using complex perpendicular orientation filters (the angle between their orientations is $\pi/2$) instead of complex quadrature filters, to obtain phase contours. A modification is then proposed to ConvNet architectures to allow it to learn complex perpendicular orientation filers, and classify patterns in the gradient of phase. Learned filters can be 2D (spatial) for 2D ConvNets, or 3D (spatiotemporal) for 3D ConvNets.

This model involving phase information is robust. It works well with contrast variations in images, and more importantly with affine deformations (caused by prospective projection) that often occur to people or objects in scenes [9]. Modeling these deformations can be important in describing actions, especially if the recognition system relies on understanding and modeling how these actions affect the surroundings, like the work done in [36]. A visual example of what a phase-contour may look like can be seen in Fig. 1d; it shows the subtraction of two phase contours. These phase contours are for the Fast Fourier Transform (FFT) basis functions, as will be explained in section 6.

3.2. Gabor Filters

An important filter that exhibits constant phase properties, and can be tuned to different band-pass frequencies, is the Gabor filter. It is a Gaussian function modulated by a sine wave. A real-valued 2D Gabor is a 2D Gaussian kernel modulated by a plane wave;

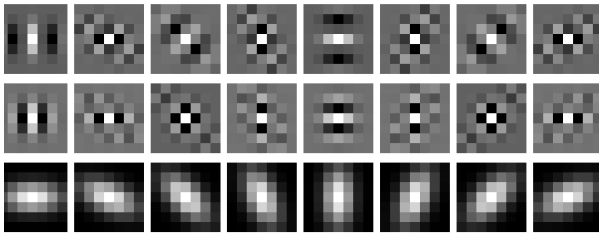


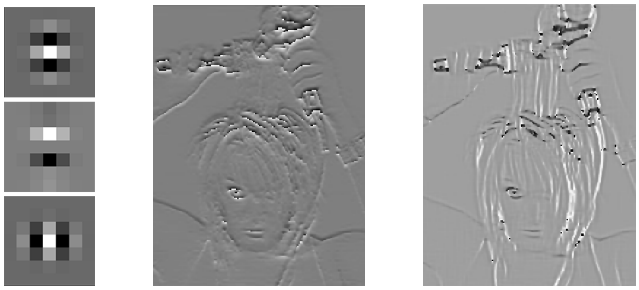
Figure 3: **A real-valued Gabor bank.** With $\sigma = 2$, $\gamma = 2$, $f = \{\sqrt{3},\ 1/\sqrt{3},\ 0.01\}$, and $\theta = \frac{\pi}{8} \times \{0, 1, .., 7\}$

$$\text{Gabor}(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\big(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\big) \times \cos\big(2\pi\frac{x'}{\lambda} + \psi\big) \quad (5)$$

where the first term represents the Gaussian kernel, and the second term represents the plane wave, with x' = x $\cos(\theta)$+y $\sin(\theta)$, and y' = $-$x $\sin(\theta)$+y $\cos(\theta)$. In the wave term, λ is the wavelength of the wave, θ represents its orientation, and ψ is its phase offset. σ is the standard deviation of the Gaussian, and γ is the spatial aspect ratio of the Gaussian, used to control its ellipticity. A complex-valued Gabor filter can then be constructed from two identical real-valued Gabor filters in quadrature *i.e*. with a $\pi/2$ phase shift separating both;

$$\text{ComplexGabor}(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \text{Gabor}(.., \psi, ..) + i\ \text{Gabor}(.., \pi/2 - \psi, ..) = \exp\big(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\big) \times\ \exp\big(i(2\pi\frac{x'}{\lambda} + \psi)\big) \quad (6)$$

Complex Gabor filters are commonly used for phase-based motion estimation [9, 11]. An example of a complex pair of Gabor filters can be seen in Fig.4a. The real part of a Gabor function is even, and is sensitive to oriented lines, while the imaginary part is odd, and is sensitive to oriented edges. The magnitude of the complex response combines the sensitivity of both components of the filter into a phase-independent measure of the orientation strength. The phase of the complex response highlights the existence of structures matching filter's orientation, frequency, and scale tuning, and gives information about the type of such structures (lines, edges, etc) [14, 33]. Since those filters are tuned to a specific orientation, a set of filters with different orientations is required to extract a complete contour of an object, as contours consist of structures (lines, edges) in all directions.



(a) (b) (c)

Figure 4: **Quadrauture vs PO filters.** (a) From to to bottom: the real and imaginary parts of a complex Gabor, followed by rotated real Gabor. Top and mid form the complex Gabor pair, while top and bottom form a complex PO pair. (b) Phase response to a complex Gabor. (c) Complex response to a complex PO. In (b), only vertical orientations are highlighted, while both horizontal and vertical orientations are highlighted in (c), providing a more detailed description of the phase contour.

4. Learning Phase-Based Descriptions

An adaptation of phase description of motion in image sequences to ConvNet architectures is proposed. First of all, a family of complex filters, which is used instead of Gabor filters to extract phase contours, is proposed. A complex-valued ConvNet layer [31] learns these complex filters. The phase of the activation maps is then taken, and passed to the remainder of the network. the architecutre can be seen in Fig. 5. Temporal differentiation is performed on the images before passing them as inputs to the network.

4.1. Perpendicular Orientation Filters for Phase Contour Extraction

Learning complex quadrature filters using convolutional networks is difficult. This is mainly attributed to the small filter sizes in ConvNets; a very popular size for the first layer filters is 7×7, and most modern architectures do not exceed that. Hence, any operations that could be used to regularize or ensure that the phase shift between real and imaginary filters is $\pi/2$ would fail, due to inaccuracies caused by having a small frequency domain resolution. An alternative could be learning imaginary filters (for example) as a weighted sum of fixed basis functions [17], then force real filters to have the same weighted sum but of $\pi/2$ phase shifted versions of those basis. However, we follow a more straight-forward approach by proposing a different type of filters to extract phase contours.

In [9, 11], complex Gabor filters are used to extract the phase. This is optimal for their use case of velocity estimation, since each complex pair must be tuned to extract

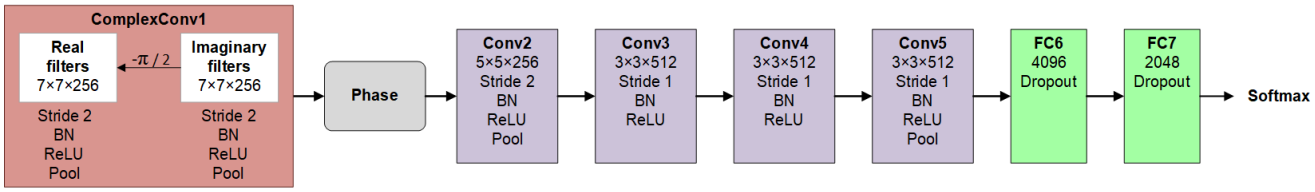


Figure 5: **Network Architecture.**

selected and unique band of measurements to properly reconstruct velocity. Any overlap with the response of the filters leads to inaccurate estimations [9, 11]. However, the context of this work is action description, and noting necessitates this need for uniqueness. Instead of using quadrature filter pairs that have the same orientation, we propose using perpendicular orientation (PO) filters. The real and imaginary pair cover perpendicular orientations (are a $\pi/2$ rotated version of each others) but are otherwise exactly identical. Each one of those filters extracts relevant information (related to its tuning) under its orientation. Combining information from two perpendicular orientations will give a more complete, but less-orientation sensitive, response from an image. We use a real-valued Gabor as basis to our perpendicular filters. The complex pair is then described as:

$$\text{POF}(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \\ \text{Gabor}(.., \theta, ..) + i\ \text{Gabor}(.., \theta + \pi/2, ..) \quad (7)$$

As with Complex Gabors, the phase of the proposed filter settings gives information about image content following the filters tuning of frequency and scale, but with a lower sensitivity to structure's orientation. This lowered orientation selectivity in turn causes the phase to describe more complete contours of the structure, as can be seen in Fig. 4. Using PO filters instead of quadrature Gabor filters cause a significant increase in performance, as will be discussed in section 6.

4.2. PO Layers: Learning Perpendicular Orientation Filters

Using nominal complex ConvNets does not guarantee learning PO filters. An alternative is to force the complex filters to be rotated versions of each others, as shown in algorithm 1. This is done by fixing real filters as a *constant* (*i.e*. not trainable) rotated version of the imaginary ones, doing one training iteration on the imaginary filters through backpropagation, then re-initializing real filters again as a constant rotated version of the new imaginary filters. This is repeated throughout training. Learning is done on the imaginary filters instead of the real ones to avoid any problems that may arise from having real filters in the denominator of the argument operator. We call complex-valued ConvNet layer that learn PO filters a *PO Layer*.

Algorithm 1 Learning complex perpendicular orientation filters

Input: Image or sequence of images
Output: Network predictions
Initialization: Initialize imaginary filters with random values
Training Process: repeat for every iteration:
1: Set real filters as a *non-trainable* version of imaginary filters, rotated by $-\pi/2$
2: Forward pass through the network
3: Calculate loss
4: Backward pass, update imaginary filter

4.3. Temporal Gradient of the Input

Since it is the temporal gradient of the phase, and not plain phase information of an image, that better represents motion, it is important to differentiate the information with respect to time. However, differentiating responses inside of a network might be troublesome. Instead, frames are differentiated before inputing them to the network. Theoretically, differentiating the response of a convolution is identical to differentiating one of the convolved functions:

$$\frac{\partial}{\partial t}(f * g) = \frac{\partial f}{\partial t} * g \quad (8)$$

provided that two conditions hold: both f and g must be absolutely integrable, and f must have an absolutely integrable (L^1) weak derivative. Input images satisfy both conditions, and it is safe to assume that learned filters satisfy at least the first; so both conditions hold here. This means that a temporally differentiated input (*i.e*. an Eulerian input) can be used and is compatible with phase-based methods. One could argue that instead of explicitly differentiating the input, the network could learn a differentiated version of the filters. However, empirical results discussed in Section 6 show the importance of input differentiation.

4.4. Sinusoidal Regularization

Since we only wish to describe motion, the constant phase criterion for motion estimation could be relaxed. It might even be better to have filters with non-constant phases, as such filters could be more useful for classification. To investigate this, a regularization method that forces filters to resemble sinusoidal wavelets, which have constant phase properties, is proposed. Performance change caused by applying this regularization will then reflect on the importance of having constant phase filters.

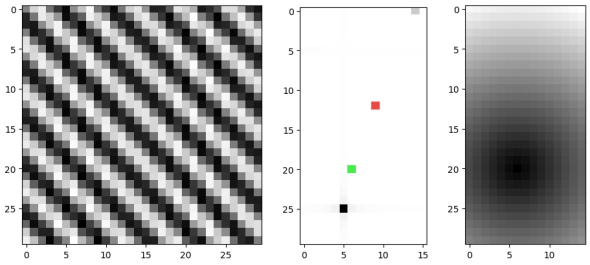
A very distinct property of sine waves is that they are represented by only one point in the magnitude domain (ignoring frequency domain symmetry of real sine waves). Any deviation from this property indicates a deviation from a pure sinusoidal form. A regularization term will be added to penalize this deviation. Calculation of this term per filter shown in algorithm 2. This regularization term forces the spread of points in magnitude domain to be low (ideally zero), while the training process moves the points around, until a sinusoidal of a certain frequency is finally obtained.

The real and imaginary filters are but a rotated version of each others, which makes it sufficient to calculate the term using only one of them. Each of these filters is represented as a real-valued matrix, and its magnitude domain will be Hermetian-symmetric, which means only half of the spectrum should be considered in the calculations. Real-FFT (RFFT) was used since it only calculates half the spectrum, which reduces required computations.

Center of Mass (CoM) of a matrix can be seen as calculating a weighted mean of the points coordinates to obtain a central frequency, which is considered the center of the spread. This ensures that the existence of spatial noise, which has a little impact spatially and a small magnitude value, does not heavily affects the location of the calculated center frequency; contrary to selecting the mean point as the center frequency. Fig. 6 shows an example using the mean vs center of mean to select the center frequency. Each point is then multiplied by its euclidean distance from the center. Points far from this frequency behave like noise, and their existence should not be over-penalized.

Algorithm 2 Regularization calculations

Input: Imaginary components of the complex PO filter
Output: Regularization loss per filter
Regularization Process: repeat for every iteration:
1: Calculate Real-FFT of the filter, get magnitude spectrum
2: Calculate the center of mass of the spectrum
3: Multiply each points by its euclidean distance from the center
4: Sum all distances to obtain the loss



(a) (b) (c)

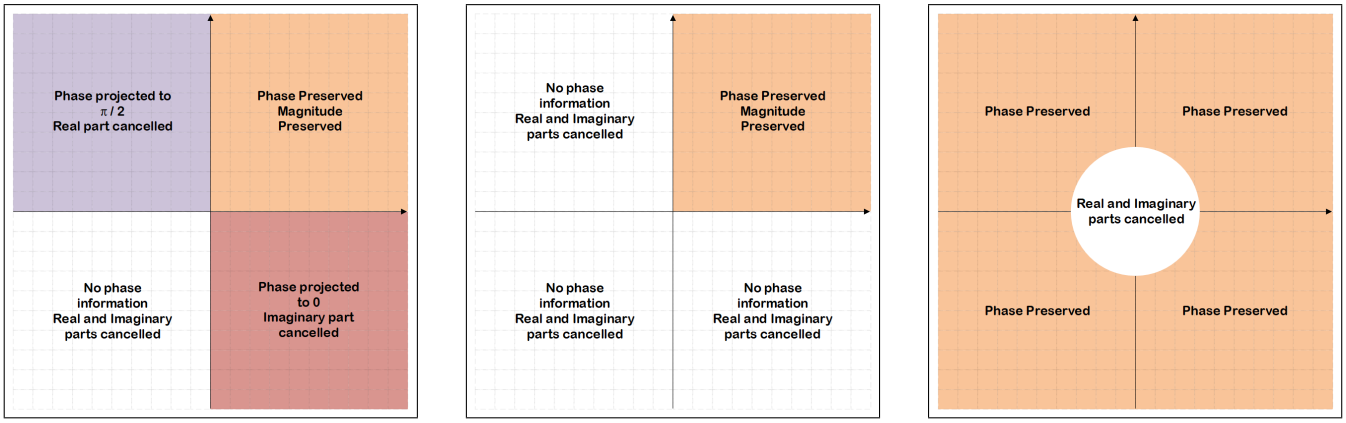
Figure 6: **Regularization to reduce spread in frequency domain.** (a) A filter consisting of a sine wave (f = 7 Hz), with additive noise (sine wave, amplitude = 0.2, f = 14 Hz). (b) Its magnitude domain showing two magnitude components for each sine in the filter, in gray-scale. The mean and the CoM, labeled in red and green respectively, show how noise shifts the center frequency in both cases. (c) The euclidean distance of each point in magnitude domain from the CoM. A Darker shade of gray indicates a higher value.

Gabor Regularization. It is possible to use this regularization technique to learn Gabor filters. To do this, two filter banks will be used for each of the imaginary and real complex filters. The first is initialized as a non-trainable Gaussian kernel with a fixed standard deviation σ, while the other will be initialized randomly. The multiplication of both filters gives the final filter, which is used for the convolutional layer. Regularization is calculated for the final filter; the Gaussian kernel effectively works as a Gaussian window for the RFFT.

Aliasing effects. Filter sizes of ConvNet layers are usually small. The maximum frequency that will correctly be represented in the magnitude domain is then half of the filter size, as Nyquist's theorem states. In our case, this causes aliasing for frequency components above 3.5 Hz. However, this will not be an issue, since the end goal is to regularize the spread, and it is highly unlikely that a filter will contain only wavelets with the same alias frequency. In all, constrains on sampling and frequency resolution can be relaxed.

4.5. Initialization

Empirically, and as real and imaginary filters have similar distributions, real and imaginary activations appear to also have similar distributions, especially with the use of batch normalization. Hence, the input ratio to the argument operator has a uniform distribution. The $atan$ function does not affect much as long as most values are within its linear range. Thus, its output distribution would also be uniform. Real-valued ConvNet layers after the argument operator should then be initialized with random uni-



(a) (b) (c)

Figure 7: **Complex Activation Functions.** (a) $\mathbb{C}$ReLU. (b) zReLU. (c) modReLU.

form values, while respecting Xavier's [12] or He's criteria [16]. Assuming that n_{in} and n_{out} are the numbers of input and output neurons respectively, random values used to initialize the weights are drawn from $[-r, r]$, where $r = \sqrt{6/(n_{in} + n_{out})}$ for Xavier's initializations, and $r = \sqrt{6/n_{in}}$ for He's. Previous work in [31] also used uniform initialization for phase values.

To initialize the complex-valued layers, the work in [31] proposes an method that helps filters learn decorrelated features. It starts with (semi-)unitary matrices that are then reshaped to the shape of the required filters. However, this initialization method caused a significant degradation in performance for our architecture. Instead, weights of the imaginary filters of the complex layers were initialized randomly following Xavier's criterion.

5. Implementation Details

Network Architecture. Network architecture can be seen in Fig. 5. It is a replica of that in [28] except the first layer which is now complex-valued, and is trained as a PO layer. Phase is calculated from complex activations, and is then passed to remaining real-valued ConvNet layers.

Complex Activation Functions. Three complex activation functions can be found in the literature: $\mathbb{C}$ReLU [31], zReLU [13], and modReLU [2]. Fig. 7 illustrates their effect on the complex domain. $\mathbb{C}$ReLU applies ReLU separately on the real and imaginary feature maps. It effectively activates neurons in quadrant 1, completely deactivates those in quadrant 3, and deactivates either the real or the imaginary parts for neurons in quadrant 2 and 4, respectively. This means that phase values of its output can remain unchanged, or be set to zero or $\pi/2$. zReLU activates only the points that fall within the 1st quadrant of the complex domain, and sets all remaining points to zero. It has one degree of freedom less than $\mathbb{C}$ReLU, since phase values of its output can either remain unchanged, or be set to zero. modReLU on the other hand, creates a "dead zone" of a certain radius b (b is a trainable variable) around the 0 origin; only neurons with values outside this circle are activated. It effectively filters points (and hence their phase information) based on the significance of their magnitude values. $\mathbb{C}$ReLU was used for all reported experiments, as experiments using zReLU and modReLU did not converge.

Applying $\mathbb{C}$ReLU to the complex feature maps, *i.e.* before calculating the phase, delivered a better performance than applying a normal-valued ReLU to the calculated phase values. This is due to the relatively higher degree of data manipulation that $\mathbb{C}$ReLU delivers.

(Complex) Batch Normalization. For the complex layer, one can choose Complex Batch Normalization (CBN) or nominal real-valued Batch Normalization (BN). In [31], using CBN was a must; all experiments using one BN on complex data failed. To standardize a complex matrix to the standard normal complex distribution, it is not sufficient to scale and translate the numbers to have 0 mean and a variance of 1, since this does not ensure that the resultant distribution is circular. CBN treats the normalization process as that of whitening 2D vectors, by scaling the data along the two principal components. Instead of this, we opted to applying BN separately to the real and the imaginary numbers. This proved to be highly effective, and empirically, both real and imaginary components had an equal variance. This also significantly reduced GPU utilization and memory consumption compared to CBN.

It is essential for all phase-based systems that the input to the argument operator falls within the linear range of

Table 1: Performance (%) of different inputs per architecture.

Input	RGB	dRGB	2-GS	dGS	dPhase	Optical Flow	5-dGS	5-dPhase	5-Optical Flow
Phase-based	51.3	48.8	49.8	74.4	70.1	-	75.3	68.23	-
VGG-M	52.3 [28]	45.5	44.2	74.3	65.4	66.7	68.75	70.8	80.4 [28]

the *atan* function, or at least away from its asymptotic region [9, 11]. The existence of BN solves this issue for ConvNet-based systems. In the performed experiments, most values fell within the required range.

Initializing PO layers from pretrained filters. To transform a ConvNet layer into a PO layer, imaginary filters are initialized form the filters of the ConvNets. Real filters are initialized as a rotated version, and the new network is trained as explained in Algorithm 1. In this case, an argument operation is further added to the network, after the PO layer. Pretraining experiments performed in section 6 are done by taking a ResNet-51 network [15], applying the mentioned process and transforming only the first convolutional layer into a PO layer.

Pretrained networks operate on RGB images, which have 3 channels (a depth of 3). Inputs like grayscale images, for example, have a depth of 1. Pretrained filters are averaged depth-wise, to transform their shape into the required one. Furthermore, if the input is a stack, filters are repeated and tiled to match the input dimension. This process is similar to that of [35].

6. Experiments and Discussion

6.1. Datasets and Experimental Setup

Datasets and evaluation protocol. Evaluation is performed on UCF101 [29], which is among the largest and most popular available annotated video datasets. UCF101 contains 13k videos, covering 101 action classes, with an average of 180 frames/video. Three standard training/testing splits are provided by the organizers. Each split contains 9.5k videos. The final evaluation reports average performance over the three splits for the dataset. The 101 actions are divided into 5 categories: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, and Sports.

Eulerian inputs. Since the size of most used action recognition datasets is relatively small, invariability to appearance is essential to avoid over-fitting. To do so, an input that is obliterated of all magnitude information is proposed. Particularity, the FFT of an image is taken, all magnitude values are set to 1 (effectively removing its information), then inverse FFT is performed. Result is called a *Phase Image*. Phase information are more important than magnitude information, as they are what defines most image details and structure [25]. Phase images proved to be very effective in minimizing over-fitting, as results show in Table 1. The temporal gradient of two phase images (the subtraction of the phase images of two subsequent frames i.e. difference of phase) is named a dPhase image. Likewise, the temporal gradient, *i.e*. the difference, of two RGB frames or grayscale (GS) frames are called dRGB and dGS respectively. Fig. 1 shows an example of dPhase and dRGB images. Invariability to appearance is the highest for dPhase images, less for dGS, and is least for dRGB.

Training and testing details. For all experiments, the network was optimized using Gradient Decent with a Momentum of 0.9. Videos are uniformly sampled from all classes to create a batch of 258; 128 per GPU. Dropout ratio of 0.9 was used for all cases. Learning rate started as 0.01, and was reduced by a factor of 10 at iterations 45000 and 75000, and was stopped at iteration 100000. All inputs were calculated on the fly. Videos frames are resized, such that the smallest side is equal to 256. A sample of size 224×224 is then randomly cropped and flipped. Samples are rescaled to (0, 255), and mean subtraction is applied.

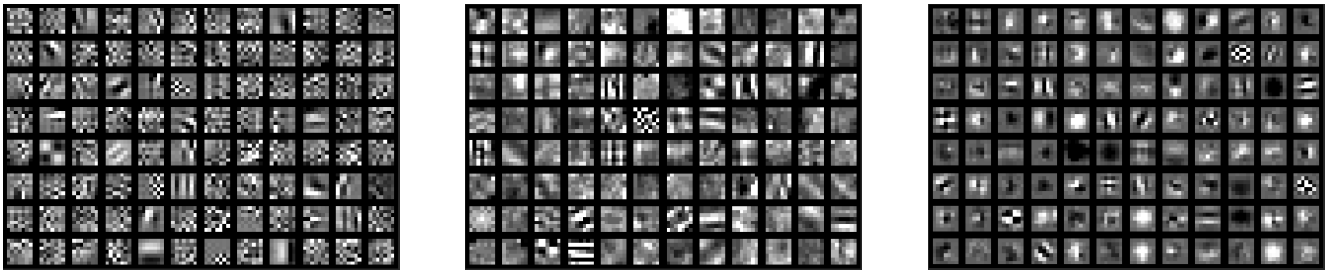
As for testing: from each video, 25 samples with uniform temporal spacing were taken. A central crop of size 244×244 is obtained from each sample. Prediction scores of those samples are then averaged to make a final class prediction for the video.

6.2. Performance Analysis

All following experiments were performed on split-1 of UCF101. Results of experiments using different inputs on different architectures are shown in Table 1. VGG-M refers to the architecture used in [28], which consists of 5 real-valued ConvNets. Phase-based refers to our proposed architecture, as shown in 5. 2-GS refers to inputting a stack of 2 GS images. Likewise for 5-dGS, and 5-Optical flow.

6.2.1 Phase-Based vs Normal ConvNets for Eulerian Representations

The Phase-based architecture delivered an increase in performance of up to 5% compared to VGG-M for all Eulerian inputs, with the exception of dGS, which delivered an equal highest-performance for both architectures. This suggests



(a) (b) (c)

Figure 8: **Visualization of learned filters.** (a) From random initializations, no regularization. (b) With Sinusoidal regularization. (c) With Gabor regularization. Contrast adjustment was applied to manifest filter details.

Table 2: Performance difference per class, for top-10 different classes, for dGS as an input. VGG-M exceed phase-based in (left), vice-versa in (right).

VGG-M	$\Delta\%$
Archery	19.5
JumpingJack	15.8
Rowing	13.9
CricketShot	12.2
Skijet	10.7
BlowDryHair	10.5
PlayingFlute	10.4
ApplyLipstick	9.4
PlayingCello	9.1
HulaHoop	8.8

Phase-based	$\Delta\%$
FloorGymnastics	19.4
TennisSwing	16.3
BoxingPunchingBag	14.3
WalkingWithDog	13.9
GolfSwing	12.8
MoppingFloor	11.8
HighJump	10.8
UnevenBars	10.7
ShavingBeard	9.3
HandstandWalking	8.8

dGS is relatively-optimum compared to dRGB and dPhase for this dataset *i.e*. includes enough invariance to not overfit, yet not decrease performance. Although both networks had an equal recognition rate for dGS, they appear to learn different things. There is an average difference of 5% in per-class recognition rate between both architectures. This difference is distributed over 40 out of the 101 classes; both architectures had an equal performance for the remaining 61 classes. Table 2 shows the top-10 classes at which each architecture outperforms the other. These differences in general can be attributed to the limitations of the phase-based methods, which relate to high-velocity motion, and noise.

Action classes like *Archery*, *JumpingJack*, and *HulaHoop*, at which VGG-M beats the phase-based network, have people or objects that move with high velocity. There is no official measures of video velocity given for this action recognition dataset, so it is difficult to quantify a "performance-velocity" curve or measure. However, same observations have been made by Guatam and van Hulle [11]. They noted a breakdown in velocity estimation performance for speeds beyond ± 5 pixels/frame. Around this range, phase gradient does not yield stable results due to phase wrap-around effects. This limitation of phase-based methods extends to our learning-based approach. Adding more PO layers increases the size of the effective receptive field *i.e*. deeper QuadNet layers will a higher scale view of actions. This in turn will decrease sensitivity to high-velocity actions. We do not investigate this further for now.

Another limitation of phase-based methods is their sensitivity to noise [9, 11, 10], as noise in input images directly translate to noise in phase measurements. This also extends to our method. Eulerian inputs of some classes at which VGG-M outperforms phase-based nets appear to be quite noisy *e.g*. splashing water in *Skijet*. This noise is caused by the temporal differentiation of input images. It translates to noise in calculated phase values from the complex filters, and hampers learning and recognition. Samel and Karam [4] investigated the effect of noisy inputs to ConvNets. They show that noise is considerably harmful for recognition, and affects activations at all layers. This can also be seen as a problem for the learning process.

But is explicit temporal differentiation of the input images at all important? Theoretically, performance of 2-GS should be comparable to that of dGS, as 2D ConvNet filters have the flexibility to learn temporally differentiating filters [28] to extract similar features from both inputs. However, this is not straightforward in practice, as indicated by the significant difference in performance in Table 1. An alternative could be using a special loss function to drive the separation of learning motion and visual information from a pair of consecutive input images, as done by Oh *et al*. [24]. Using such methods could further improve the performance of the phase-based network, as it solves the explained noise-related problem. We do not investigate that further in this work.

Importance of constant-phase filters. Results of experiments using Sinusodial and Gabor regularization are shown

in Table 3. These experiments are performed using dGS as an input. Visualization learned filters in Fig. 8 confirm that the regularization did have the effects intended.

Regularization did not lead to an increase in performance when training from scratch. This shows that while using constant-phase filters is essential for correct motion estimation, as mentioned in section 3, it is actually bad for recognition systems. The additional complexity of having non constant-phase filters is rather useful for classification.

Table 3: Results with regularization. Having non constant-phase filters helps with recognition.

Regularization Type	Regularization Strength		
	1e-3	1e-4	1e-5
Sinusoidal	72.0%	69.29%	70.24%
Gabor	66.7 %	-	-

Benchmarking. There are two main aspects to benchmark: (1) The quality and importance of learning vs hand-crafting PO filters; and (2) the performance of quadrature Gabor filters vs PO filters. The later aspect reflects on the performance of the proposed method for extracting phase contours.

To quantify the quality of learned PO filters, we initialize the PO layer with rotated real Gabor filters. Those filters are fixed throughout training. Results serve as a benchmark to the learning process. Furthermore, to compare quadrature filters with PO ones, the complex layer was initialized with quadrature Gabor filters, also fixed throughout training. Table 4 shows the results of the experiment. Two settings for the filter banks are used. The first one consists of 24 filters and is similar to the one in [9], shown in Fig. 3 (but with γ =1). The other consists of 96 filters, covering 12 logarithmically spaced frequencies between 0.2 and 5 Hertz, over the same 8 directions $\theta = \pi/8 \times \{0, 1, .., 7\}$.

Table 4: Benchmark results, using dGS as an input. PO outperforms quadrature Gabor.

Filter Type	No. of Filters	Testing Accuracy	Training Accuracy
Quadrature Gabor	24	60.5%	$\sim$90%
PO Gabor	24	64.8%	$\sim$70%
PO Gabor	96	71.6%	$\sim$80%

Some experiments involved training the filters that were initialized with Gabor banks. The end filters were not very different from the initialized ones, except being spatially stretched (equivalent to using a higher γ), and there was no notable increase in performance over random initializations. We also experimented with initializing with 3D PO Gabor filters. 3D filters of size $7 \times 7 \times 7$ were applied to a stack of 7 input frames in a 2D ConvNet setting *i.e*. there was no temporal sliding of filters during convolution. Performance was lower than that of 2D Gabor filters. This shows that even when filters are indeed 3D (thus, are spatiotemporal), it is the temporal sliding of filters in 3D Convolutions that contributes to their efficiency in extracting temporal features.

The performance increase of learned filters, compared to fixed initialization, is not as significant as expected. This clue, in addition to the considerable amount of noise in filters as seen in Fig. 8a, confirms that noise in Eulerian inputs is a main contributor to the potential loss in performance. Gaussian Blur did not have a significant impact on reducing this noise, or else the Gabor regularization would have performed better than Sinusodial regularization (Table 3). Median filtering, which is quite effective with this type of noise, could damage fine details at the borders of objects and humans, which are potentially important for action recognition [26]. A viable alternative is to learn the Eulerian transformation from a sequence of consecutive frames instead of manual temporal differentiation, as discussed above.

As for quadrature v PO filter: PO filter had a non-trivial increase in performance. This can be attributed to giving the network more information about phase contours per-filter, which helped better extract information in the early layer and pass it to later layers. This increase is in favor of PO filter. However, this does not rule out quadrature filters as a future direction for phase-based approaches, as performance could vary for different network sizes, architectures, and applications (*e.g*. velocity estimation).

6.2.2 Eulerian vs Lagrangian for Action Recognition

This comparison was done between optical flow trained on VGG-M, and dGS using the phase-based architecture. In all, 60 out of the 101 classes had a difference in per-class performance. Actions that belonged to the Playing Sports category had the biggest variation in performance. Eulerian outperformed on actions like *Shotput*, *PoleVault*, *FieldHockeyPenatly*, and *FloorGymnastics*, while optical flow outperformed on *HandstandPushups*, *BaseballPitch*, and *BodyWeight*. There is no specific pattern that describes this behavior when it comes to the action itself. However, a considerable amount of videos for actions at which Eulerian outperformed had dense crowds in the background. Optical flow estimations for such backgrounds is extremely challenging [1], are noisy for the actions in this dataset, and appears to affect the classification process. Furthermore, actions like *Haircut*, *BlowDryHair*, *Trampoline* and *PlayingDaf*, are challenging to calculate the optical flow for, since they include smooth surfaces or objects that are hard to track *e.g*. hair. As expected, Eulerian outperformed on

such actions. We did not notice any patterns relating to the effects of global motion on any of the inputs. This could be due to applying mean subtraction during the preprocessing for both inputs.

Theoretically, for water sport actions like *Rowing* and *Skiijet*, optical flow models water motion poorly, so there could be an increase in performance for Eulerian inputs. However, optical flow performance on such actions was similar to that of dGS on VGG-M. The phase-based network failed to capitalize on Eulerian inputs for such actions due to the discussed problem with noise.

When it comes to performance, testing accuracy using a single dGS image exceeded that of optical flow. However, stacking dGS or dPhase lead to only a slight increase in recognition rate. This is unfavorable of dGS, compared to the significant increase in performance gained by stacking optical flow. This is mainly attributed to lack of data. In addition, just as discussed in section 2.1, using 3D ConvNets is essential to good extraction of temporal information. This will be revisited in future work, due to the current limitation in available computational resources.

In general, the behavior of Eulerian representations appear to be in between that of RGB and optical flow. It encodes motion information and has similar training behavior, *i.e*. loss curve, as optical flow. Yet, stacking it leads to a trivial improvement in performance, similar to that of stacking RGB [28]. Based on these observation, using Eulerian could potentially have a significant increase in performance using 3D ConvNets, and continue to outperform optical flow, just like RGB.

Contribution of motion information. To quantify how motion information in a specific representation contribute to classification, video frames are shuffled before calculating the input, effectively removing the temporal structure of the original video [26, 37]. The relative change in testing accuracy reflects the contribution of motion of that input using that architecture, as shown in Table 5. Optical flow numbers taken from [26] for the TSN architecture [35]. TSN processes optical flow in stacks of 5. In all, motion contribution is comparable for all inputs. Appearance remains the largest contributor to performance. The phase-based architecture had a comparable use of motion information for the dGS input, compared to the VGG-M network. However, as seen in Table 1, it did for other types of inputs.

Table 5: Relative contribution of motion ($\Delta\%$) of each input type. All inputs are generally comparable.

Input	dGS	dGS	Optical flow
Architecture	Phase-based	VGG-M	TSN
Normal	74.4%	74.3%	86.9%
Shuffled	51.4%	49.4%	59.6%
Contribution of motion	30.1%	33.5%	31.4%

6.3. Comparison with the State-of-the-Art

Pretraining. Table 6 shows performance of training using an ImageNet pre-trained ResNet-51 network. It was trained using an initial learning schedule of 1e-2, that is decreased by a factor of 10 at iterations 10k and 20k. Training is stopped after 40k iterations. No Dropout was applied, as it is not traditionally used in ResNet networks [15]. Pretraining lead to a significant decrease in training time, but not to an increase in performance. In fact, testing accuracy is slightly less than that of training from scratch on the phase-based architecture. This is also the case for optical flow [7]. Not using Dropout contributes significantly to the loss in performance [7]. However, ResNets have significantly fewer parameters and require less computational resources to train. This makes the performance compromise more plausible. A counter-intuitive note is that dRGB did not benefit from pretraining. This cannot be attributed to the nature of the phase-based model and its focus on learning different things than spatial models, since GS-related inputs benefited from pretraining. It is rather the problem with RGB-related inputs; the challenge of having a small dataset, and not being able to use dropout in this case.

Table 6: Performance (%) of the transformed pretrained ResNet-51.

Input	dGS	5-dGS	dRGB	5-dRGB
Accuracy	72.8%	70.8%	41.1%	44.1 %

Comparison of results with other networks that have a motion representation input is shown in Table 7. Two points to keep in mind while comparing is the complexity and configuration of the implemented network, and the size of its input.

TSN Networks are an ensemble of 3 two-stream networks, and provide several architecture-level improvement over them, so are not directly comparable. However, even with advanced techniques, pretraining, and a $\times 6$ bigger number of inputs, only 8% improvement was obtained.

When it comes to two-stream networks [28], two-stream ResNets [7], and motion vectors [38], the temporal input is a stack of 10. As discussed, we have exceeded the performance of 1 flow frame (Table 1), but stacking flow frames gives a bigger improvement than stacking Eulerian inputs with 2D ConvNets. This is the reason for the 5% difference in performance with the flow-based networks. Still, a difference of 5% is quite comparable, and reflects on the potential of using such inputs in more advanced architectures. On

Table 7: Comparison of performance (%) over UCF-101. Comparison of temporal streams is the focus of this work.

Network	Two-Stream Network [28]	Two-Stream ResNets [7]	TSN Networks [35]	Motion Vectors [38]	ActionFlow Networks [22]	Proposed Phase-Based
Input(s)	10-Optical flow + 1-RGB	10-Optical flow + 1-RGB	3 × (10-dRGB + 1-RGB)	10-Enhanced Motion vectors	2-RGB	5-dGS
Spatial	72.8 %	82.3 %	84.5 %	N/A	-	-
Temporal	81.2 %	79.1 %	83.8 %	79.3 %	70.0%	75.3 %
Two-Stream	88.0 %	89.47 %	87.3 %	86.4 %	-	-

the other hand, ActionFlowNetworks [22] learn to generate optical-flow specific for action recognition. Performance of such networks is still far from that of the calculated optical flow. They have also tried using 16-RGB instead of 2-RGB as an input, and obtained 83.9 %. This remains a moderate improvement over manually calculated optical flow, especially given the difference in input size.

In all, stacking Eulerian did not outperform stacking Lagrangian representations. However, 3D ConvNets hold a great potential for Eulerian representations, considering that they did cause a significant improvements for stacked RGB inputs, and the similarities in the training behavior of Eulerian with RGB.

6.4. Limitations

Limitations of learning phase-based descriptions is similar to those of classic phase-based approaches, namely dealing with noisy inputs and high-velocity actions. Learning an Eulerian transformation from two consecutive frames could potentially solve the former problem, while increasing the number of PO layers could solve the later. Another limitation to keep in mind is the number of distinct motions per neighborhood. Having 3 or more motions per neighborhood in a video (which is unlikely to happen [9]) increases the possibility of motion description errors. This could happen if the effective size of the receptive field of a PO layer is too high. Images in Fig. 1c, of the *HulaHoop* and *Skijet* actions, are viable examples of high-velocity actions and noisy temporal inputs respectively.

7. Conclusion and Future Directions

We present a new architecture to learn phase-based descriptions from Eulerian inputs, in the context of action recognition. This included a method to learn perpendicular orientation filters using complex-valued ConvNet layers, which we called PO layers. Empirical analysis showed that this architecture delivers an improvement for several Eulerian inputs, while also exceeding the baseline for recognition using a single optical flow input. The proposed temporal model learns features that compliment those extracted from appearance models (operating on RGB images), and can be used in a two-stream setting to boost performance. A thorough empirical analysis of the performance of Eulerian inputs compared to optical flow was presented, in addition to an analysis of pros and cons of the proposed phase-based method, compared to normal 2D ConvNets.

In future, we would like to improve this model in several ways. First of all, to capitalize on the fact that Eulerian inputs compliment spatial inputs (*i.e*. RGB), two-stream fusion [8, 7] is likely to significantly improve performance. Second of all, capturing long-term temporal structures is important to recognition in general. This can be done by further fusion from several temporal samples [8] or by incorporating our proposed system in an LSTM network [5, 23]. Finally and most importantly, we believe using 3D ConvNets architectures [3] will significantly increase recognition rate, especially when applied to datasets with sufficient size and video variability [20]. Future research directions that could be inspird by our trainable phase-based approach include, but are not limited to, problems like learning optical flow estimation [6], and action localization [27].

References

[1] S. Ali and M. Shah. Floor fields for tracking in high density crowd scenes. In *European conference on computer vision*, pages 1–14. Springer, 2008.

[2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464, 2015.

[3] J. Carreira and A. Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[4] S. Dodge and L. Karam. Understanding how image quality affects deep neural networks. In *Quality of Multimedia Experience (QoMEX), 2016 Eighth International Conference on*, pages 1–6. IEEE, 2016.

[5] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[6] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks.

In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[7] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal residual networks for video action recognition. *CoRR*, abs/1611.02155, 2016.

[8] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. *CoRR*, abs/1604.06573, 2016.

[9] D. J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *Int. J. Comput. Vision*, 5(1):77–104, Sept. 1990.

[10] D. J. Fleet, A. D. Jepson, and M. R. Jenkin. Phase-based disparity measurement. *CVGIP: Image Understanding*, 53(2):198 – 210, 1991.

[11] T. Gautama and M. A. V. Hulle. A phase-based approach to the estimation of the optical flow field using spatial filtering. *IEEE Transactions on Neural Networks*, 13(5):1127–1136, Sep 2002.

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

[13] N. Guberman. On complex valued convolutional neural networks. *CoRR*, abs/1602.09046, 2016.

[14] L. Haglund and D. J. Fleet. Stable estimation of image orientation. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 3, pages 68–72. IEEE, 1994.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society.

[17] J.-H. Jacobsen, J. van Gemert, Z. Lou, and A. W. Smeulders. Structured receptive fields in cnns. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2610–2619. IEEE, 2016.

[18] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3192–3199. IEEE, 2013.

[19] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[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. *CoRR*, abs/1705.06950, 2017.

[21] S. Meyer, A. Djelouah, B. McWilliams, A. Sorkine-Hornung, M. Gross, and C. Schroers. PhaseNet for Video Frame Interpolation. *ArXiv e-prints*, Apr. 2018.

[22] J. Y. Ng, J. Choi, J. Neumann, and L. S. Davis. Actionflownet: Learning motion representation for action recognition. *CoRR*, abs/1612.03052, 2016.

[23] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4694–4702. IEEE, 2015.

[24] T.-H. Oh, R. Jaroensri, C. Kim, M. Elgharib, F. Durand, W. T. Freeman, and W. Matusik. Learning-based Video Motion Magnification. *ArXiv e-prints*, Apr. 2018.

[25] A. V. Oppenheim and J. S. Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5):529–541, May 1981.

[26] L. Sevilla-Lara, Y. Liao, F. Güney, V. Jampani, A. Geiger, and M. J. Black. On the integration of optical flow and action recognition. *CoRR*, abs/1712.08416, 2017.

[27] Z. Shou, D. Wang, and S.-F. Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1049–1058, 2016.

[28] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[29] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[31] C. Trabelsi, O. Bilaniuk, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal. Deep complex networks. *CoRR*, abs/1705.09792, 2017.

[32] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

[33] M. Van Ginkel, P. Verbeek, and L. Van Vliet. Multi-orientation estimation: Selectivity and localization. In *3rd Annual Conference of the Advanced School for Computing and Imaging, Heijen, NL, June 2-4*, 1997.

[34] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman. Phase-based video motion processing. *ACM Trans. Graph.*, 32(4):80:1–80:10, July 2013.

[35] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool. Temporal segment networks: Towards good practices for deep action recognition. *CoRR*, abs/1608.00859, 2016.

[36] X. Wang, A. Farhadi, and A. Gupta. Actions ˜ transformations. *CoRR*, abs/1512.00795, 2015.

[37] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy. Rethinking spatiotemporal feature learning for video understanding. *CoRR*, abs/1712.04851, 2017.

[38] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang. Real-time action recognition with enhanced motion vector cnns. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2718–2726. IEEE, 2016.

[39] Y. Zhu, Z. Lan, S. Newsam, and A. G. Hauptmann. Hidden two-stream convolutional networks for action recognition. *arXiv preprint arXiv:1704.00389*, 2017.

Appendices

A

Background

This chapter provides an overview of concepts needed to understand the work presented in the scientific paper. It stars with an introduction to ideas in image processing, then proceeds to cover the main building blocks of Convolutional Neural Networks (ConvNets), and complex-valued ConvNets. The section on image processing focuses explains the convolution operation and how it can be used for filtering, in addition to complex filters. It also gives an introduction to optical flow, and how it calculates motion trajectories from two consecutive frames. The section on ConvNets cover the following building blocks: ConvNet layers, activation functions, batch normalization, loss functions, and regularization.

A.1. Ideas in Image Processing

A.1.1. Convolutions

A convolution is an integral function that computes the amount of overlap between two functions [1]. It as expressed using the following equation:

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{A.1}$$

Specifically, this integral calculates the overlap between functions f and g as g is shifted over f by τ. A visual expression of the operation is shown in Fig A.1. It is widely used in for filtering in image processing, and signal processing in general [1].

A.1.2. Filtering, and Complex Filters

Images can be filtered by convolving the image with a filter. Filters are also called kernels in literature. The response of an image to the filter varies depending on the used filter. Fig A.2 shows an example of using an averaging filter over an image. There exist filters for blurring, sharpening, edge detection, and more [3]. Gabor filters are an example of more advanced filters. They are characterized by a sine wave modulated by a Gaussian kernel. A real-valued 2D Gabor is a 2D Gaussian kernel modulated by a plane wave [1], as follows;

$$\text{Gabor}(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \times \cos\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{A.2}$$

where the first term represents the Gaussian kernel, and the second term represents the plane wave, with $x' = x\ \cos(\theta) + y\ \sin(\theta)$, and $y' = -x\ \sin(\theta) + y\ \cos(\theta)$. In the wave term, λ is the wavelength of the wave, θ represents its orientation, and ψ is its phase offset. σ is the standard deviation of the Gaussian, and γ is the spatial aspect ratio of the Gaussian, used to control its ellipticity. Gabor filters are widely used for texture analysis, as they can extract spatial information that fall within a specified range of frequencies, with the determined orientation. Fig A.3 shows an example of a 2D Gabor filter, along with a demonstration of a Gabor filter applied to a Chinese character. The result of the filter can be passed to

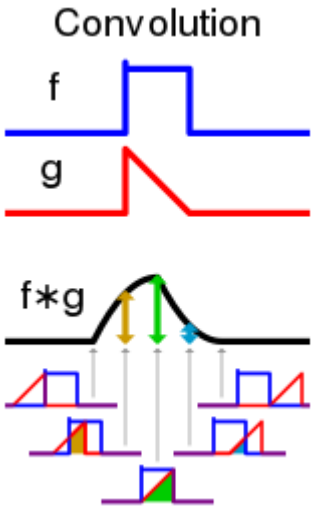


Figure A.1: **A visual explanation of the convolution operation [2].** As a flipped version of g is slided over f, the overlap between both functions is calculated, and expressed as a value of the output function $f * g$.

a classification system that uses the existence of certain features with the filter's specified frequency and orientation to classify the class of the character.

In many applications, it is important to get the phase and magnitude of some filtered input for analytical purposes [6]. For example, in this work, the phase of a filtered sequence of images is observed to detect its variations, and use those variations to describe motion in the sequence, and recognize the action in it. Many areas deal with complex filters, like microscopy, astronomy, and optical imaging [6]. A significant amount of this usage is focused on measuring phase information, as they hold most information in an image [6]. Fig A.4 demonstrates the importance of phase.

The related work of the scientific paper in Chapter 1 discusses several applications that use complex filters, like motion magnification [7] and velocity estimation [8]. Those applications are all phase-based. Some of them (e.g. motion magnification [7]) decompose the image using a bank of complex filters with different scales and orientations. The result of this decomposition is called a complex steerable pyramid [9]. It is complex because it was obtained using complex filters, thus each response in it is complex. It is called a *pyramid*, because filters of different scales were used in the decomposition, and the scale of the responses form what looks like a pyramid. *Steerable* refers to the directionality of the responses in the decomposed representation. Complex filters are often *Quadrature*, which means that their imaginary and real parts have a $-\pi/2$ phase-shift between them [1].

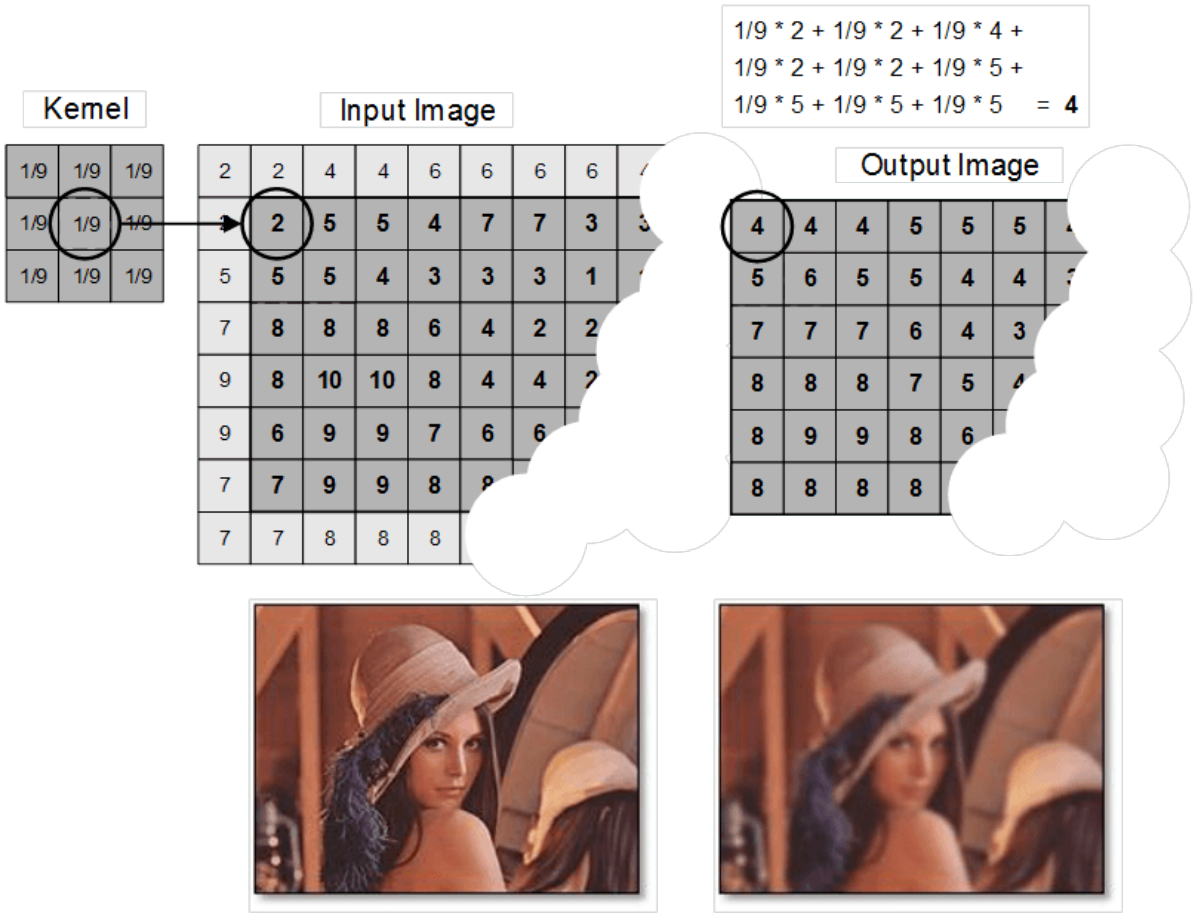
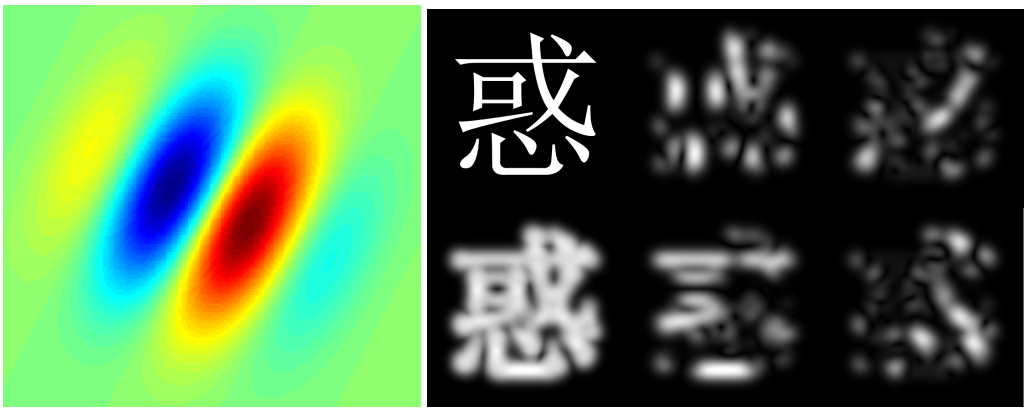


Figure A.2: **A visual explanation of image filtering [3].** As the 2D averaging kernel slides over the image, its values are multiplied point-wise with the image. The result is the summation of all multiplication. This result is a value of the output image. The complete output image is the result of convolving the kernel over the complete image.



(a) (b)

Figure A.3: **The use of Gabor filters for texture analysis.** (a) A 2D Gabor filter, magnified. Red refers to a positive value, while blue refers to a negative value [4]. (b) (Top left) a Chinese character; (Bottom left) The super-position of applying different Gabor filters with several different orientations to the Chinese character. Results from each individual filter are shown in middle and right [5].

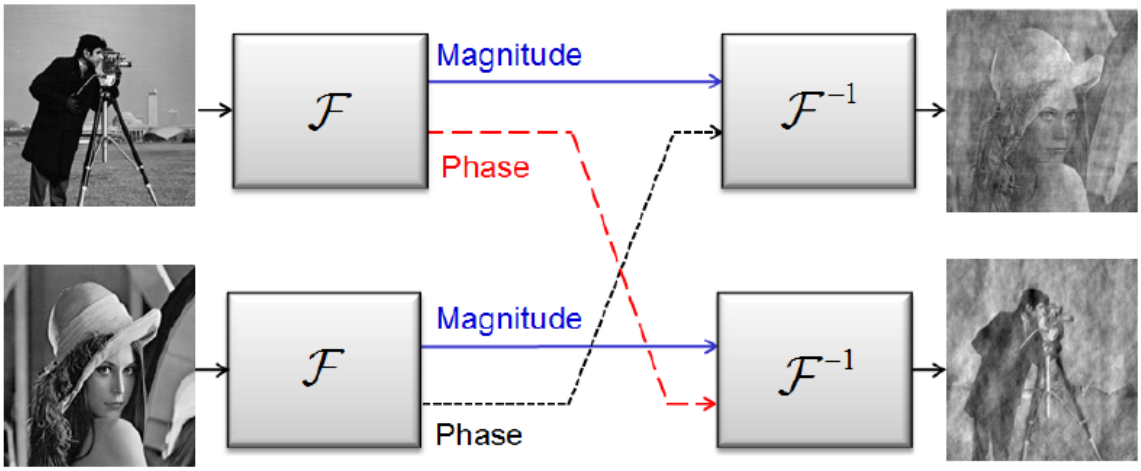


Figure A.4: **The importance of phase [6].** The Fourier Transform is calculated for two images. Applying Inverse Fourier Transform after swapping the phase of the images clearly shows that most information are encoded in phase.

A.1.3. Optical Flow

Optical flow (OF) describes the pattern of apparent motion in a scene using trajectory information. Trajectory information are obtained by tracking image properties between two consecutive frames. When a difference in location of a property (e.g. a corner) is detected, a vector from the old location in frame 0, leading to the new location in frame 1, encodes the trajectory information. Figure A.5 shows an example. OF estimation can be *dense* when all pixels in a sequence of images are tracked. It also can be *sparse*, when only key features (corners, textures) are tracked. The questions that different OF methods approach differently are (1) what image properties should be tracked? and (2) how to track it? [8] Regardless of the method, it is often assumed that pixel intensities of an object do not change between consecutive frames (brightness consistency). Hence, the general OF equation can be describe as:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{A.3}$$

Assuming movement is small, and using Taylor series, an approximation of the equation can be developed to:

$$I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{A.4}$$

From this, it follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{A.5}$$

Which yields:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{A.6}$$

Where V_x, V_y are component velocities in the x and y directions respectively [8]. To solve this equation with two unknowns, another set of of equations is needed. Different methods have different assumptions to do so. An example can be given using the most two famous OF estimation methods, which are the Lucas-Kanade method [8, 11], and the Horn-Schunck method [8, 12].

The Lucas-Kanade method [11] assumes that the displacement of points between two nearby frames is small, and all neighboring pixels have similar motion. From this, the optical flow equation is assumed to hold for all pixels within a window. The obtained set of equations is then solved using least squares. The Horn-Schunck algorithm [12] assumes smoothness in the flow over the whole image. From this assumption, distortion of flow over the whole image is minimized [8]. Practically, assumptions like these break e.g. variation in lighting conditions, sharp motions, etc [8]. This yield incorrect optical flow estimations. This work knowledges these problems, and tries to solve it by using a different motion representations as inputs for classifiers.

A.2. Building Blocks for Convolutional Neural Networks

A typical ConvNet architecture consists of a number of concatenated blocks, where each block contains a Convolutional layer, followed by an activation function –optionally, it can also include a pooling layer, and a batch normalization layer [13, 14]. The higher the number of these blocks, the *deeper* is an architecture. [14]. These concatenated blocks extract features from images and videos. They are followed by fully convolutional layers, which predict the classification label from a combination of these extracted features [13, 14]. Figure A.6 gives an example of a typical ConvNet architecture.

A.2.1. Convolutional Layers

The idea of using filters to extract features out of images was explained in Section 2.1.2. When designing filters, a question always stands out: what makes a certain filter better suited for a this task than other filters? Convolutional layers try to avoid asking this question, by learning to extract features optimized to the task at hand, instead of using pre-defined filters [13, 14]. This layer learns filters of shape ($size \times size \times depth$), where each point in the filter is

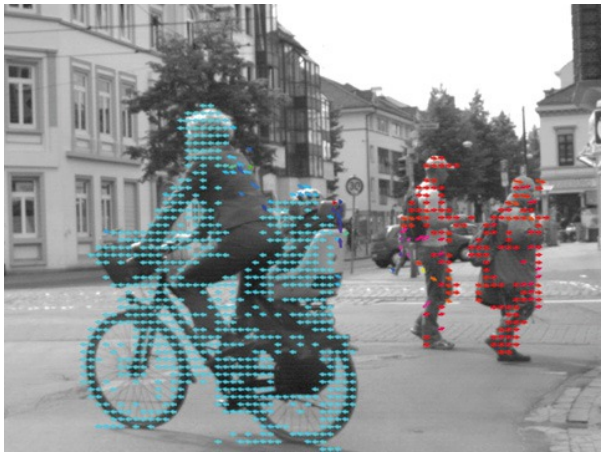


Figure A.5: **An estimated optical flow field superimposed on its related frame [10].** .

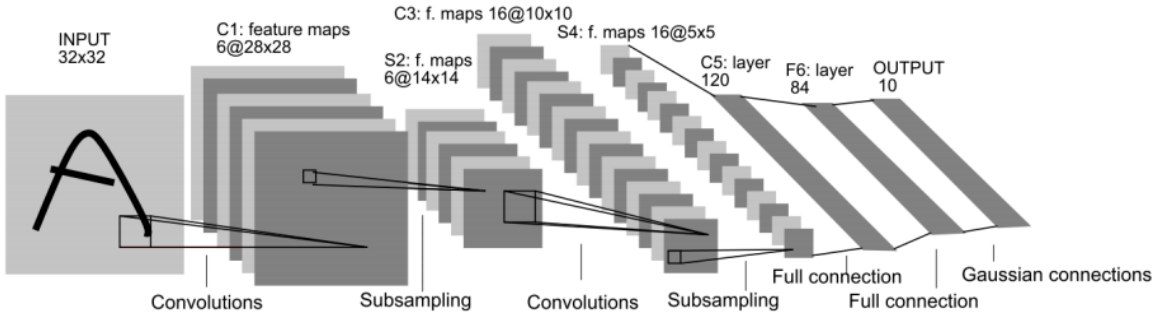


Figure A.6: **The LeNet ConvNet architecture for character recognition [13].**

a neuron. They are applied to inputs of shape ($height \times width \times depth$). The output shape is then ($height \times width$). A specified number of filters is used. The overall output of the layer is then of shape ($height \times width \times number\ of\ filters$). The output of the layer is called a *Feature Map*. The third dimension, which is the number of filters, is the ”depth” of the input to the following layer [15]. Figure A.7a shows an example of the operation.

In ConvNets, often pooling layers are used to reduce the dimensionality, thus the number of neurons in the network [13–15]. This controls the number of parameters in a network, and is done by moving a window of a certain size, and computing a down-sampling function on that window. Most popular types are average pooling and max pooling, where the output of the window is the either the average or the maximum size of that window.

A.2.2. Activation Functions

Each neuron in a convolutional layer has an activation functions. Most popular activation functions are non-linear [16, 17]. Thus, they allow neural networks to compute nontrivial problems using only a small number of nodes [17]. The input to an activation function is a feature map. Outputs are called *Activation maps*. Activation functions are essential to performance. Networks that do not use them are rare [13, 14, 16].

A.2.3. Fully Connected Layers, Loss Functions, and Regularization

Fully connected layers are basically neural networks, where each neuron is connected to all neurons in the previous layer [13–15]. These networks are concatenated after the activations of the last convolutional layer, and receive all values from it. Typically, two or three fully connected networks are used, where the size of the last networks is equal to that of the number of predicted classes, and its output gives the prediction scores (sometimes called Logits) for that input. The last layer is followed by a *Softmax* layer [18]; a normalization layer that rescales all value such that their summation equals to 1. Softmax outputs the estimated

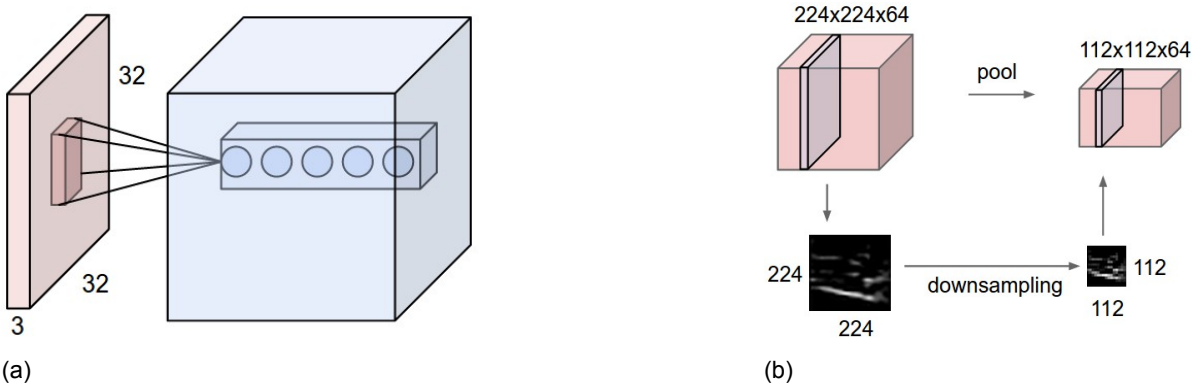


Figure A.7: **(a) A convolutional layer [15].** The input is an RGB image of size $(32 \times 32 \times 3)$. Filters have the same depth as the input. The number of filters is assumed to be 5. Thus, the output volume has a depth of 5. **(b) An example of pooling [15].** The pooling window size is (2×2).

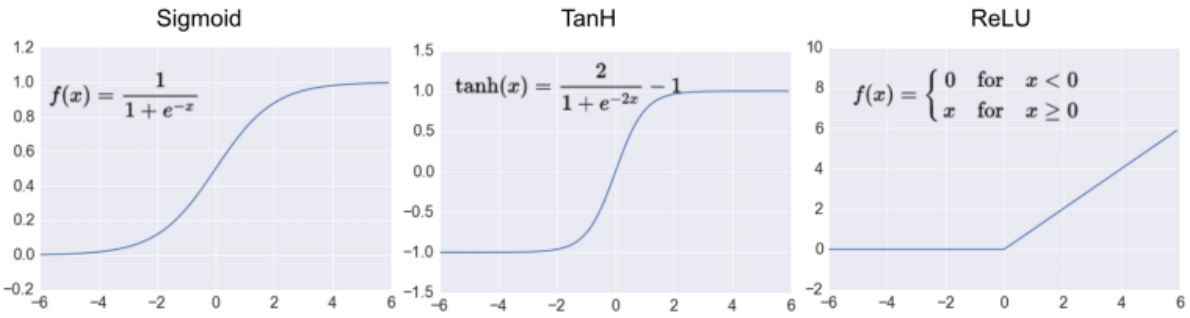


Figure A.8: **Commonly used activation functions [17].**

prediction probabilities. The index of the largest value corresponds to the predicted class label [15, 18].

Ideally, softmax output for an input image should be equal 1 at the index corresponding to that class, and 0 elsewhere. Any difference from that behavior should be penalized. This penalty is calculated using a loss function. The loss function takes at least the softmax probabilities and the true class label as an input, and outputs a loss value corresponding to the difference between those inputs; the bigger the difference, the bigger the loss. This loss value is used to optimize the network using backpropagation [15].

Often, the size of the network is quite large as a big number of parameters is required to properly learn huge datasets. However, this often leads to overfitting [15]. Hence, regularization is very important to avoid overfitting. One of the most used forms of regularization is Dropout [15, 19]. When dropout is applied to a certain layer, neurons in this layer are randomly dropped (based on a predefined drop probability) during backpropagation [20]. Dropped neurons do not receive gradient updates. This was found to prune the neurons in the network, and is very effective against overfitting [15]. Regularization can also be done by adding a special loss term (multiplied by a regularization strength λ) to the loss function [21]. This enforces the network adapt to the goal of the regularization function. For example, it is common to use the L1 and L2 distances [21] the weights as a loss function, to force coherence in the weight value and avoid relatively large values in a filter.

A.2.4. Batch Normalization

The batch normalization layer [22] is typically inserted after a convolutional layer and before the activation function. This layer zero-centers the feature maps and normalizes them by the standard deviation, so it outputs modified feature maps with unit-Gaussian distribution. This makes convergence of networks much faster, and the network convergence and

performance less dependent on initializations [15, 22].

Background Bibliography

[1] Bracewell, R. N., & Bracewell, R. N. (1986). The Fourier transform and its applications (Vol. 31999). New York: McGraw-Hill.

[2] A visual comparison of convolution, cross-correlation, and auto-correlation. By Cmglee, from Wikimedia Commons.

[3] Heterogeneous compute case study: image convolution filtering. Retrieved from https://www.imgtec.com/blog/heterogeneous-compute-case-study-image-convolution-filtering/

[4] 2D Gabor filter. By English Wikipedia user Joe pharos, from Wikimedia Commons.

[5] Gabor filtering for Chinese OCR. By Johndoe154 [Public domain], from Wikimedia Commons.

[6] Shechtman, Y., Eldar, Y. C., Cohen, O., Chapman, H. N., Miao, J., & Segev, M. (2015). Phase Retrieval with Application to Optical Imaging. IEEE Signal Processing Magazine, 32(3), 87-109.

[7] Wadhwa, N., Rubinstein, M., Durand, F., & Freeman, W. T. (2013). Phase-based video motion processing. ACM Transactions on Graphics (TOG), 32(4), 80.

[8] Fleet, D., & Weiss, Y. (2005). Optical Flow Estimation. Mathematical Models in Computer Vision: The Handbook.

[9] Portilla, J., & Simoncelli, E. P. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. International journal of computer vision, 40(1), 49-70.

[10] Determination of localized motion in live video data. By commonvisionblox.com

[11] Lucas, B. D., & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision.

[12] Horn, B. K., & Schunck, B. G. (1981). Determining optical flow. Artificial intelligence, 17(1-3), 185-203.

[13] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[14] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[15] Stanford University (2016). Lecture notes for CS231 Convolutional Neural Networks for Visual Recognition. Retrieved from http://cs231n.github.io/

[16] Jain, A. K., Mao, J.,& Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. Computer, 29(3), 31-44.

[17] KDNuggets. Neural Network Foundations, Explained: Activation Functions.

[18] Hinton, G. E., & Salakhutdinov, R. R. (2009). Replicated softmax: an undirected topic model. In Advances in neural information processing systems (pp. 1607-1614).

[19] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[20] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560.

[21] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[22] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.