POLITECNICO DI TORINO

Master Degree in Computer Engineering

Master Thesis

EGO-T^3

Test Time Training for Egocentric videos

Supervisor
Prof.ssa Barbara Caputo

Co-supervisors:
Dott. Mirco Planamente
Dott.ssa Chiara Plizzari

Candidate
Simone Alberto Peirone

October 2022

Abstract

In the last few years, the technological advancement of wearable cameras has led to an increasing interest in egocentric (first-person) vision. The ability to capture activities from the user's perspective has provided significant opportunities for a more in-depth study of human behavior compared to the third-person setting, as sensors are much closer to actions and embed a natural form of attention that stems from the human gaze direction. The research community highly benefited from egocentric vision for a variety of different tasks, such as human-object interaction, action prediction and anticipation, wearer pose estimation, and video anonymization.

A crucial aspect for several video-related tasks is their multimodal nature. Audio, RGB, and optical flow provide complementary insights that are critical to a thorough understanding of the real world. In contrast, continuous head movement, variations in lighting conditions and differences in the way humans complete the same task represent a source of bias that strengthens the coupling between the model's predictions and the training domain, affecting its ability to generalize to unknown environments. Several Domain Adaptation (DA) techniques have been proposed to make models more robust. Among these, Unsupervised Domain Adaptation (UDA) combines labeled source data and unlabeled target data to reduce the distance of the extracted features across different domains. However, real-world applications require more flexibility, as target samples are often scarce, unrepresentative or even private, limiting the applicability of UDA. Test Time Training (TTT) appears to be a viable solution to these issues, with domain adaptation performed directly at test time under the simple assumption that input samples provide clues on the actual distribution of the target domain which could be used to improve predictions.

With TTT, models undergo multiple adaptation steps at test time by minimizing an adaptation loss on target data and updating normalization statistics. This work provides, for the first time, a comparative analysis of multiple adaptation techniques on the EPIC-KITCHENS dataset. Particular attention was given to the analysis of their dependence on batch normalization layers and the impact of repeated adaptation steps, two critical concerns for real-time and power-constrained applications. Experiments indicate strong accuracy improvements, with up to 3.6% (absolute) gain over several baselines across a variety of settings, suggesting that TTT effectively improves model performance in the presence of dynamic environments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The past few years have seen an exciting growth of deep learning, which has revolutionized our approaches to countless fields, from more classical tasks such as image classification, natural language processing, and recommendation systems to the astonishing recent achievements in image generation from textual prompts. As models have become increasingly complex, with billions of parameters requiring large data sets for training, a new trend has emerged to move deep learning solutions closer to the end users. In our everyday life, the smartphone keyboard predicts the next word based on the current sentence we are writing, the speech recognition models of our home assistants allow us to seamlessly control the lights in our homes, and we can edit photos to remove unwanted parts or add filters with just a tap.

Computer vision is one of the fields that has benefited the most from recent advances in deep learning, with remarkable advancements in a range of tasks from image classification, object detection and image segmentation to video-related tasks such as classification, detection and anticipation of actions. The application of deep learning techniques to videos has proven to be particularly challenging for at least three main reasons. First of all, videos combine a *spatial* and a *temporal dimension* that are essential for a better understanding of what is happening in the video, in addition to its visual content. Joint learning from both is a crucial issue for several video-related tasks. Second, video incorporates multiple sources of information. RGB frames capture the visual aspect of the environment, audio detects sound events and optical flow shows the relative movement of elements in the scene. Each of these and other sources provides particular insights that may or may not be crucial, depending on the task, although learning from multiple modalities has proven to be a difficult task [5, 1]. Finally, the computational costs associated with video processing and the scarcity of high-quality datasets have been a huge obstacle for the research community even though the situation drastically improved in the last decade.

Figure 1.1: Camera setting used to collect the Epic-Kitchens dataset [6]. The wearable camera points directly at the action in progress, just like the gaze of its wearer.

As far as video-related tasks are concerned, research has long focused on third-person vision, that is, video in which the camera is external to the recorded action, mainly because of the difficulty of collecting first-person video due to the bulky equipment required. The availability of wearable cameras has dramatically improved this situation, making first-person video recording much more accessible and leading to the birth of *egocentric vision*. Head-mounted cameras record from a point of view that coincides with that of the wearer, providing a view of the actions that are taking place that is strongly affected by the movement of the operator's head. Because humans tend to focus their gaze on what they are doing and on the position of their hands [7], camera movement provides an attention mechanism that continuously follows the unfolding of the action, unlike third-person videos in which the point of view is fixed and the action itself moves within the scene, possibly occluded by other objects or humans. The change in perspective between third-person and first-person video has a dramatic impact on the ability of deep learning models to better understand the environment and the way humans interact with it, a crucial step to bring unprecedented solutions in everyday life. Egocentric vision can foster the development of new assistive technologies for human beings, such as real-time audio descriptions of the world to support a visually impaired person or to remind a person with short-term memory impairment where they left their keys.

The very dynamic nature of egocentric videos highlights a problem, known as *domain shift*, common to countless fields of deep learning, namely the inherent difficulty of models to adapt to different domains. This problem stems from the

fact that models learn to solve tasks directly from data, which can lead them to incorporate knowledge that is completely irrelevant to the task at hand, degrading their ability to generalise. As far as action recognition is concerned, humans are able to recognise a person running, regardless of whether he or she is on an athletic field or in a gym. The same does not apply to neural networks. If during training the model was only trained on videos of people running outdoors, it might not recognise a person running indoors, simply because it has learnt to look for an outdoor environment to classify the action.

Egocentric vision not only suffers from the domain shift from these macroscopic changes in the environment, but also from the *continuous variations in the background* of the captured actions, which constitute a further source of confusion in the model's predictions. Several approaches have been proposed to bridge the gap between different domains. Most of these techniques operate in the training phase, either by stimulating the model to better generalise across multiple training domains, or by learning better representations for a specific target domain.

A recent emerging trend is represented by *Test Time Training* (TTT), which exploits samples from the target domain to tune the model parameters by minimizing an adaptation loss. With TTT the entire adaptation process is postponed to the test phase. *Test Time Adaptation* (TTA) is a more rigorous version of Test Time Training, which precludes access to training data as the adaptation process must be based exclusively on available target data.

1.1 Research goal and contributions

This work explores the applicability of Test Time Adaptation (TTA) techniques to egocentric vision, and in particular to improve the performance of a First Person Action Recognition (FPAR) task (see Sec. 3). All experiments were performed on the popular Epic-Kitchens dataset [8]. The aim is to investigate whether TTA is well-suited to address the dynamic nature of egocentric vision, with a focus on analysing what is the main enabler of the most common adaptation approaches, as well as one of its most noticeable limitations, namely the dependence of the adaptation process on the statistics collected by batch normalization layers. Experiments are performed over different modalities to assess how much each modality suffers from the domain shift problem and whether or not TTA can be useful in improving performance. Particular attention is devoted to the analysis of multiple adaptation step.

Finally, the last contribution lays the groundwork for further development of TTA techniques beyond FPAR, with a focus on *sim2real* adaptation of the optical flow estimated by a state-of-the-art network.

Chapter 2

Deep Learning

This chapter and its subsections provide an overview of modern deep learning, paying particular attention to the aspects most closely related to our work. First, Section 2.1 presents the motivations that led to the development of deep learning. Section 2.2 covers the fundamental concepts behind neural networks, starting from the perceptron model and addressing several aspects of the training process. Then, sections 2.3 extends the theory on neural network with visual applications.

2.1 Introduction

Deep learning arose from the idea of replicating the majestic processes that regulate the human brain. The first efforts to model the behaviour of networks of neurons with computational circuits date back nearly a century, with the theoretical work of McCulloch and Pitts (1943) [9] and the first implementation by Frank Rosenblatt (1958).

Conventional machine learning and statistical approaches, like Logistic Regression (LR) or Support Vector Machines (SVM), are often described as shallow methods, given their simple architectures consisting of a few computational layers that provide limited modelling capability. Indeed, learning in high-dimensional spaces poses a difficult challenge for shallow methods. To address this issue, shallow methods rely on feature engineering to extract meaningful information from raw data before applying any learning process. However, the cost of feature engineering becomes prohibitively high as the dimensionality increases and data becomes more sparse. The problem becomes particularly relevant in the context of vision applications, as images and videos consist of billions of pixels with little correlation between them. A single pixel in an image has little or no statistical significance but is part of local and global structures that provide insights into the image content. It is not surprising that the popularity of deep learning stems, at least initially, from

applications related to image classification, a notable example being the model proposed by Yann LeCun for handwritten zip code recognition [10]. The goal of deep learning is to learn directly complex structures and patterns directly from data.

2.2 Neural networks

Biological neural networks are sparse interconnections of neurons that exchange information through electrical signals. Each neuron receives electrical impulses from its neighbours in the network and propagates an electrical impulse if the inputs are above a certain threshold. Similarly, artificial neural networks are large circuits of simple units, the *so-called* artificial neurons, that mimic the interconnections of their biological counterparts. Each neuron computes a linear combination of its inputs and propagates an output value to all its neighbouring neurons. The similarity between biological and artificial neural networks is mostly conceptual, as the high-level mechanisms governing the human brain are still poorly understood.

The term deep refers to the layout of the network as neurons are typically organized in several layers. Tens or hundreds of layers form a computational graph between inputs and outputs, a significant difference from shallow methods that allow neural networks to discover high-level concepts directly from low-level features, e.g. the pixel of an image, by stacking multiple layers of knowledge. From a theoretical point of view, the large scale of the computational graph enables neural networks to act as universal function approximators [11]. As a result, neural networks have been successfully used in various applications, from speech recognition to image synthesis, thanks to their high flexibility and little dependence on the peculiarity of the application domain.

2.2.1 The artificial neuron

Biological neurons connect to external stimuli or other neurons through tiny filaments called *dendrites* which receive inputs in the form of electrical signals. If the voltage potential reaches a certain threshold, the neuron discharges by firing an electrochemical pulse through its *axon*. An artificial neuron is the mathematical analogue of the biological neuron and models the same mechanism by computing a linear combination of its inputs, plus a bias term, followed by an activation function that decides whether the neuron should be active or not.

The first prototype of an artificial neuron was proposed by Frank Rosenblatt in 1958, under the name of *Perceptron* [12]. Potentiometers were employed to implement the input weights while a simple threshold function acted as the activation function, similarly to its biological counterpart. Equation 2.1 translates in mathematical terms the function implemented by the Perceptron, where x_i and w_i are respectively the *i-th* input and its associated weight, b is a bias term and σ_τ is the

threshold function (Eq. 2.2).

$$y = f(\mathbf{x}) = \sigma_\tau(\mathbf{w}^T\mathbf{x} + b) = \sigma_\tau\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

$$\sigma_\tau(y) = \begin{cases} y \geq \tau & 1 \\ y < \tau & 0 \end{cases} \tag{2.2}$$

Neurons can be composed to implement arbitrarily complex functions. The non-linearity introduced by the activation function allows neurons to represent nonlinear functions. If the activation function were not present, neurons would be able to implement only linear functions and their composition would also be linear. In addition to the σ_τ, several activation functions have been proposed over the years. Section 2.2.5 provides more details on the most commonly used ones.

2.2.2 Feed-forward neural networks

To extend the limited modelling capability of a single neuron, multiple neurons sharing the same inputs can be stacked to form a *so-called* layer which is defined the vector mapping $f_\theta(\mathbf{x})$.

$$f_\theta(\mathbf{x}) = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b}) \tag{2.3}$$

A *Feed-Forward Neural Network*, also known as *Multi-Layer Perceptron*, is a sequence of multiple layers, forming an arbitrarily deep computational path between inputs and outputs. Data flows from the first layer, the *input layer*, through a series of *hidden layers* before reaching the *output layer*. Overall, the network defines a mapping which is the composition of the mappings of each layer.

$$\mathbf{y} = (f_{\theta_1} \circ f_{\theta_2} \circ f_{\theta_3})(\mathbf{x}) \tag{2.4}$$

The parameters θ of the network are tuned using an iterative optimization approach to minimize a loss function computed on the output of the last layer of the network.

2.2.3 Loss functions

Conceptually, loss functions measure the fitness of the neural network to solve a certain task. The objective of the training process is to find the parameters θ^* which minimize the loss function over the training data.

$$\theta^* = \arg\min_\theta \mathcal{L}(x; \theta) \tag{2.5}$$

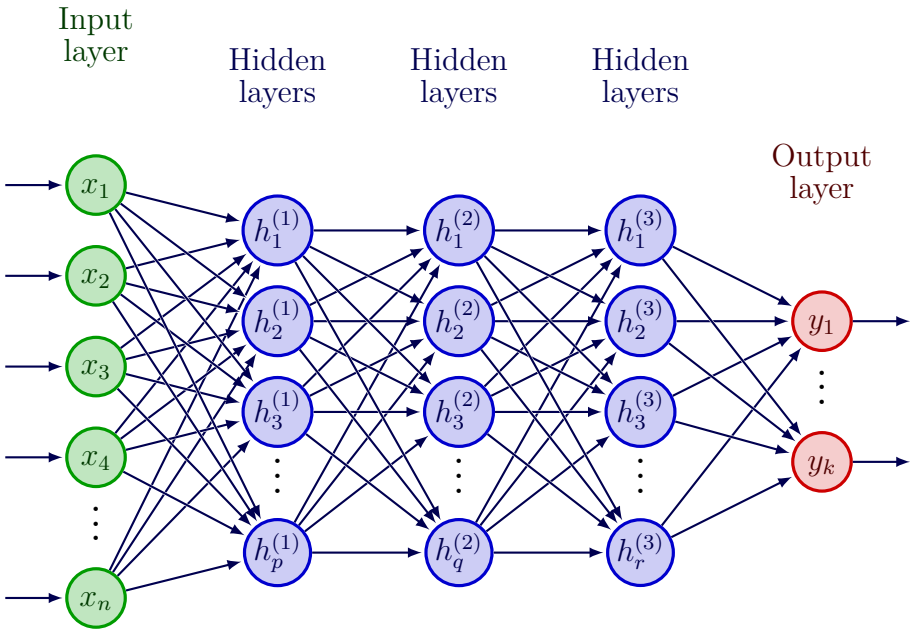


Figure 2.1: Architecture of a Feed-Forward Neural Network. Data flows from the first layer, through multiple hidden layers, each with a possibly different number of neurons, to the final output layer.

Cross Entropy loss A network attempting to solve a classification problem must associate data samples $\mathbf{x}$ with the corresponding label y. Namely, the network estimates a posterior probability distribution $p(\mathbf{y}|\mathbf{x};\theta)$. By default, the last layer of the network outputs raw values that lack a probabilistic interpretation. To address this issue, a softmax function is usually applied to the outputs to normalize the values into a probability distribution. Note that the exponential function in the softmax formulation gives much more importance to the high values of y_i. Ideally, the output of the softmax function should have a high peak in correspondence of correct sample label and be almost flat at all other points, indicating that the network is very confident in its label assignment.

$$\mathbf{p}_c = \sigma(\mathbf{y})_c = \frac{e^{y_c}}{\sum_{c=1}^{C} e^{y_c}} \tag{2.6}$$

The *cross entropy* loss function evaluates the distance between the model predictions and the true posterior distribution $\hat{\mathbf{y}} \sim p(\mathbf{y}|\mathbf{x})$, by computing their cross entropy between the two. $\hat{\mathbf{y}}$ is described by a categorical random variable whose

entry $\hat{y}_i$ is 1 if i is the correct sample label, 0 otherwise.

$$\ell_{CE}(\mathbf{p}, \hat{\mathbf{p}}) = \mathcal{H}(\mathbf{p}, \hat{\mathbf{p}}) = -\sum_{c=1}^{C} y_c \log \hat{p}_c \tag{2.7}$$

2.2.4 The learning process

Optimization problems are usually phrased in terms of an objective function $J(\theta)$ to be minimized or maximized with respect to the models parameters θ. Typically, the objective function is the average of a loss function L over the training set.

$$J(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim p(x,y)} L(\mathbf{x}, y; \theta) = \frac{1}{m} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, y^{(i)}; \theta) \tag{2.8}$$

The objective function $J(\theta)$ is known as the *empirical risk* of the model. The term *empirical* highlights that the joint distribution of samples and labels at training time $p(x, y)$ may differ from the distribution of the test data $\tilde{p}(x, y)$. The objective is minimized only on the training data, under the assumption that reducing the empirical risk might also reduce the risk on the test partition, also known as the *true risk*.

If the problem formulation allows formal guarantees about the convexity of the objective function, then it is solvable in closed form, as in logistic regression or SVMs. In general, neural networks are highly nonlinear, which makes this approach not applicable. Gradient descent is an algorithm for solving optimization problems when no guarantees on the objective function landscape are available. Gradient descent updates the model parameters by moving in the direction of the negative gradient (Fig. 2.2) to reach the minima. Given the current estimate of the model parameters θ^t, gradient descent computes the objective function and its gradient at θ^t and derives the new parameters θ^{t+1}, as in Eq. 2.9. The training process ends once the magnitude of the steps falls below a certain threshold.

$$\theta^{t+1} = \theta^t - \gamma \nabla J(\mathbf{x}, y; \theta^t) \tag{2.9}$$

The *learning rate* γ controls the size of the gradient descent steps and is a critical hyper-parameter. A small learning rate slows down the optimization process, while a high value may prevent the gradient descent algorithm from reaching a minimum. A popular technique consists of using a rather high learning rate during the early stages of the training process and gradually reducing it using a decay function. Note that the performance of gradient descent depends on the starting point θ^0 and may reach a solution that is not globally optimal.

Stochastic Gradient Descent Computing the objective function over the entire training set is often computationally unfeasible. Instead, Stochastic Gradient

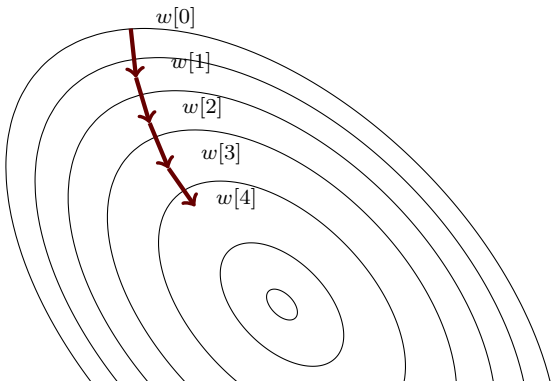


Figure 2.2: Gradient descent. The gradient is orthogonal to the contour lines of the function. Following the direction of the negative gradients, gradient descent moves towards the minimum point.

Descent (SGD) is a variant of the gradient descent algorithm that computes the objective function and its gradient on randomly sampled portions of the dataset, called *mini-batches*. Given a mini-batch of m samples, the parameters are updated according to:

$$\theta^{t+1} = \theta^t - \gamma \left(\frac{1}{m} \sum_{i=1}^{m} \nabla L(\mathbf{x}^{(i)}, y^{(i)}; \theta^t) \right) \tag{2.10}$$

SGD greatly speeds up the learning process at the cost of a reduced accuracy in gradient estimation. Somewhat surprisingly, the noise introduced by the sampling process may also have a regularization effect and help escape local solutions.

2.2.5 Activation functions

Activation functions introduce non-linearities into the computational graph of neural networks and are a key aspect of their modeling capability. The choice of the activation function directly affects how the neurons are involved in the training process. Indeed, the derivative of the activation function controls the propagation of the gradient flowing from the output to the input of the neuron. If the derivative is zero, the gradient flow is interrupted and the neuron does not learn anything.

Binary Step As previously discussed, the binary step function outputs a constant 1 if its inputs grows above a certain threshold, usually zero. The derivative of the function is always zero, which prevents learning through gradient updates.

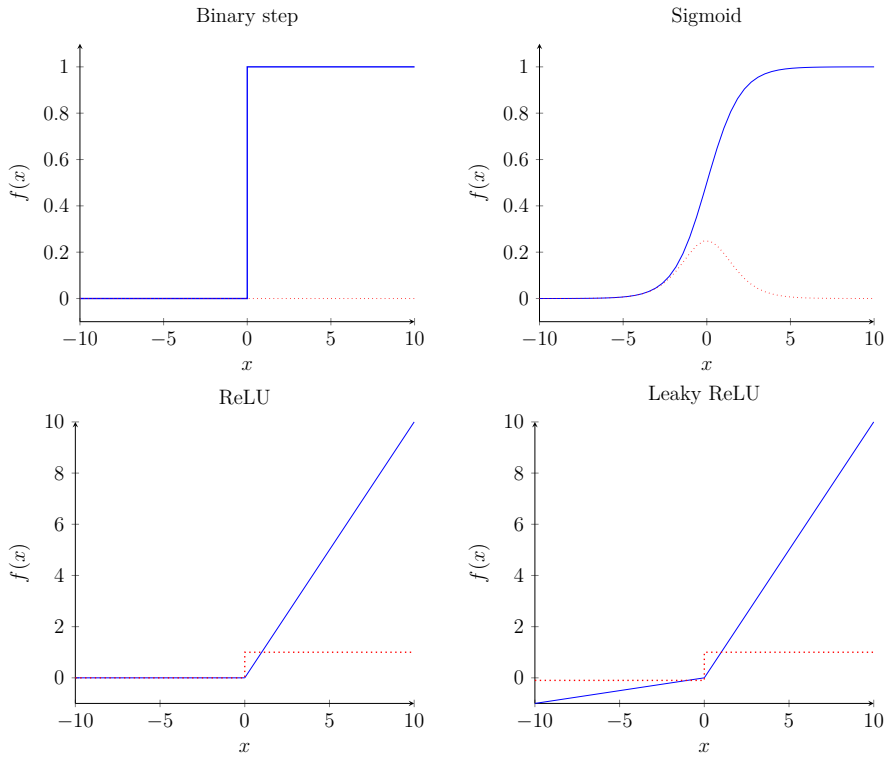


Figure 2.3: Common activation functions. The plots show four popular activation functions, in blue, and their derivatives, in red

Therefore, the binary step function is almost never used in practice.

$$\mathrm{H}(x) = \begin{cases} 1 & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{2.11}$$

Sigmoid The *Sigmoid* is a smooth function that maps the input in the interval [0,1], with two horizontal asymptotes as $x \to \pm\infty$. Its derivative has significant magnitude around zero and vanishes quickly. Therefore, the interval in which the neuron learns from its inputs is limited and the gradient flow is blocked when the input falls outside the active region around zero. Also, the derivative is always positive, indicating that the gradient updates have the same sign of the incoming gradient, limiting the parameters range explored by the neuron during the training process.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

ReLU The *Rectified Linear Unit* (ReLU) is a very popular activation function in deep learning. It implements the identity function for positive inputs and is zero in

all everywhere else. The ReLU function is unbounded and the derivative is positive and constant in the right half of its domain.

$$\text{ReLU}(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{2.13}$$

Leaky ReLU The zero derivative of ReLU in the negative half of its domain prevents learning in that region. Leaky ReLU adds a small linear contribution in the negative region to enable the neuron to learn. Also, the derivative of the Leaky ReLU function has both positive and negative regions, improving the learning process.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0.1x & \text{for } x < 0 \end{cases} \tag{2.14}$$

2.2.6 Regularization

Models usually access only a portion of the dataset during training which may lead the model to exhibit excellent performance on training data and drop sharply on the unseen samples. A model that *over-fits* the training data captures not only meaningful information but also noise and residual information that is irrelevant to the task. Regularization describes a set of techniques that improve the generalization ability of machine learning models, from very mathematically rigorous gradient penalties to more empirical approaches that aim to introduce noise into the training process with the goal of preventing or reducing overfitting.

Norm penalties

L2 regularization adds the norm of the model parameters θ to the objective function of the model.

$$\begin{aligned} \tilde{J}(\theta) &= J(\theta) + \frac{\lambda}{2}||\theta||_2^2 \\ \nabla_\theta \tilde{J}(\theta) &= \nabla_\theta J(\theta) + \lambda\theta \end{aligned} \tag{2.15}$$

Equation 2.15 show the updated objective function and its gradient. When computing the weights update, the contribution of the L2 norm progressively shifts the weights towards the origin. The regularization factor *measures* the complexity of the solution represented by the parameters and drives the models towards simpler hypothesis. In other words, smaller weights assign less importance to the individual sample features, requiring the network to focus more on significant features and less on the noise.

Dropout

Dropout [13] sets to zero randomly selected neurons or connections between different neurons with probability p, forcing the network to develop redundant paths and improving its resistance to noise. Conceptually, training a model with dropout is equivalent to training a large ensemble of smaller networks, each consisting of the neurons retained at each iteration. Dropout is typically applied after the fully connected layers of the network and is usually active only at training time. To ensure that the activations computed at training and test time have similar magnitudes, the latter are scaled by the probability factor p to account for the different number of active neurons at each step.

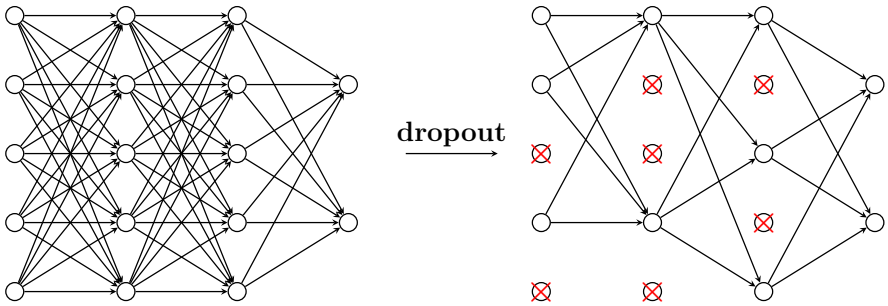


Figure 2.4: Dropout

Data augmentations

A common technique to reduce overfitting, especially in the presence of a small training dataset, consists in creating fake samples from the real training data by applying various augmentations, from affine transformations to noise injection and colour manipulation. Data augmentations effectively increase the model invariance to small alterations of the input, improving generalization.

2.2.7 Normalization layers

The deep nature of neural networks has undoubtedly proved to be very effective in solving complex problems. Although powerful, this architecture results in generally unstable inputs at each layer, as small variations in the deepest layers add up quickly and are amplified in the shallower layers. This problem is known as *internal covariance shift* and becomes even more relevant when using Stochastic Gradient Descent as the inputs change at each training iteration. Smaller learning rates and regularization techniques may mitigate the problem at the cost of a slower training process or worse performance.

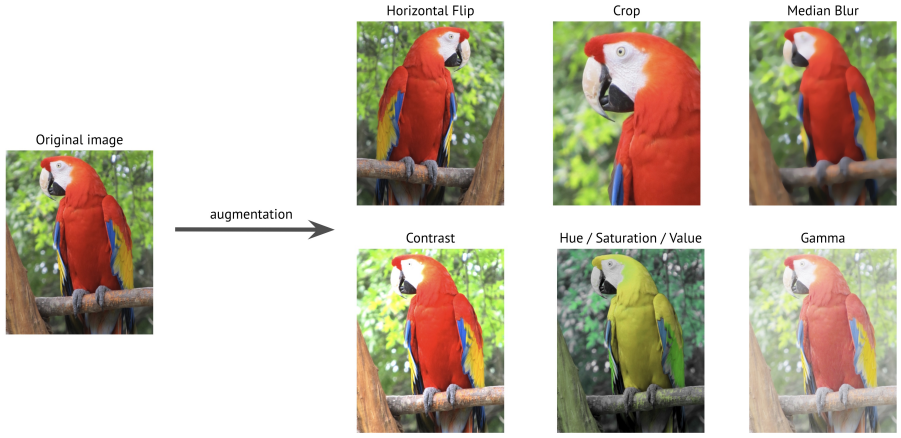


Figure 2.5: Common data augmentations applied to an RGB image. Images from [14].

Batch Normalization

Batch Normalization (BN) [15] normalizes layer outputs to a Gaussian distribution, ensuring similar input distributions among different batches of data. Batch normalization can be incorporated into neural networks like any other layer, and its behavior varies depending on whether the network is in training or evaluation mode. In training mode, the BN layer computes the empirical mean and variance of its input data $\mathbf{x}$. The mean is subtracted from the input and the result is divided by the variance to make it Gaussian distributed. Finally, two learnable parameters, γ and β, allow the network to undo the transformation introduced by the normalization layer, if the gradient updates point the network in that direction.

$$\mu_{\mathcal{B}} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma_{\mathcal{B}} = \sqrt{\frac{1}{m}\sum_{i=1}^{n}(x_i - \mu_{\mathcal{B}})^2} \tag{2.16}$$

$$y_i = \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}\gamma + \beta \tag{2.17}$$

The network keeps running estimates of the mean $\tilde{\mu}$ and standard deviation $\tilde{\sigma}$ statistics computed over the training data. During testing, the running estimates are used in place of the batch statistics since the underlying data distribution of test data is not known.

$$\tilde{\mu} = (1-\lambda)\tilde{\mu} + \lambda\mu_{\mathcal{B}} \qquad \tilde{\sigma}^2 = (1-\lambda)\tilde{\sigma}^2 + \sigma_{\mathcal{B}}^2 \tag{2.18}$$

The authors of [15] shows the integration of batch normalization into existing networks dramatically speeds up the training process by allowing the use of higher learning rates. Also, batch normalization improves the resilience of the network with respect to weights initialization.

Other normalization layers

Batch normalization depends on large batch sizes to compute accurate estimates of the mean and variance of the training data. Motivated by this limitation, several alternatives have been proposed able to work on single samples. Layer normalization [16] transforms each input channel of each sample separately. Group Normalization [17] generalize the approach of Layer Normalization by operating on groups of channels.

2.3 Convolutional Neural Networks

Theoretically, MLPs are capable of modelling any function [11], but they lack any sort of spatial or temporal awareness over the processed data. Images have a 2d structure that makes neighbouring pixels highly correlated one to the other. The relative position of pixels is critical to understanding the actual content of the image. Neural networks may or may not discover this out of sight information during training, creating stronger connections between close pixels. A more clever approach is to embed this *prior knowledge* on the input structure inside the network architecture. Convolution Neural Networks (CNN) are a subset of neural networks that include convolutional layers to learn local features from the input.

CNNs have been a milestone in the development of deep learning, from Yann LeCun's early work [10] to the drastic advances of recent years that led to above human-level performances in a wide variety of computer vision tasks.

2.3.1 Convolutional layer

Convolution is a mathematical operator commonly used in signal processing and computer vision to express how a function *interacts* with another function by sliding over it. Formally, the convolution of two real-valued functions f and g is expressed as:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.19}$$

where the integral variable τ moves function g over all the real axis. A similar equation describes convolution in the discrete domain:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{2.20}$$

In computer vision, convolution is widely used to implement several image filters, from sharpening to blur addition and edge detection, and is based on discrete convolution. A fixed size matrix, known as the *kernel*, slides over the image pixels and, for each position of the kernel relative to the input image, pixels are weighted

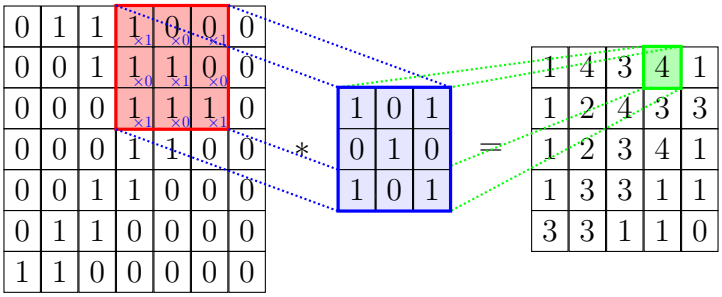


Figure 2.6: 2D Convolution. The values in the red box (matrix on the left) are weighted according to the kernel values (matrix in the middle) and mapped to the output (matrix on the right).

according to the values of the kernel and summed together. Figure 3.7 shows an example of a 2D convolution on a matrix.

RGB images are equivalent to a stack of 2d matrices, one for each color channel. 2D convolution can be easily extended to support RGB images by making the kernel three dimensional as well. A convolutional layer typically embeds multiple filters that are applied independently on the input data. Starting from different initial weights, each filter can learn different local characteristics of the input. Outputs are then piled together as if they were color channels (*feature map*). The number of filters is referred to as the *depth* of the convolutional layer.

As shown in Fig. 3.7, the output is computed only when the two matrices, the input and the kernel, fully overlap. As a result, the size of the output is smaller than the input, and some boundary information may be lost in the process. Adding padding around the input avoids losing boundary information and preserves the input size. Sometimes, a smaller output may be helpful to compress the information, learn higher level features and reduce the effect of noise. The *stride* parameter controls the step size of the sliding kernel. A stride greater than 1 results in a smaller output, since some positions of the kernel are skipped. Remarkably, the stride can take values less than 1. In this case, the operation is referred to as *fractionally-strided convolution* or *deconvolution*. The main use case of deconvolution is learning an upsampling function for input features, such as mapping an image to a higher resolution.

Another significant improvement brought by convolutional layers over fully connected layer is weights sharing. The same kernel slides over the entire image, unlike MLPs where the size of the weights matrix scales with the number of inputs and outputs.

Receptive field The receptive field of a neural network identifies the input region involved in computing an output feature; for example, a 3x3 kernel summarizes nine values to produce each output feature. The cascading convolutional neural network extends the receptive field when the receptive fields of input features are further combined to compute the output. In other words, the receptive field defines how far each feature is able to see. Larger kernels result in larger receptive fields, but the associated computational cost grows rapidly.

2.3.2 Pooling layers

Pooling levels divide the input into several smaller patches and reduce each patch to a single value using a synthesis function, such as max or avg. Depending on the patch size, the result is a dramatic down-sampling of the input that reduces the computational cost and memory usage of successive layers of the network and improves model invariance with respect to small transformations. In fact, if a feature is slightly translated, it is likely to still be part of the same patch, leaving the output of the pooling layer unaffected.

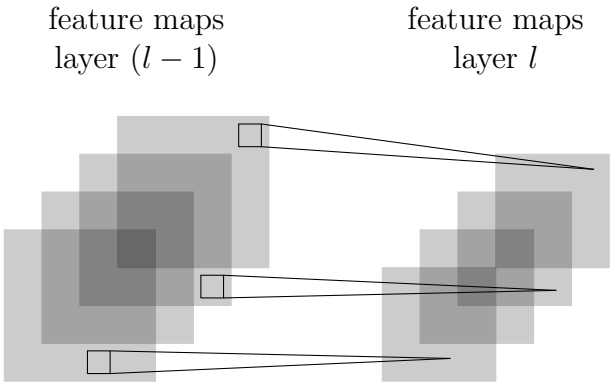


Figure 2.7: A pooling layer. Each square patch from the input feature maps (left) is reduced to a single value of the output (right) using a synthesis function.

Common pooling functions compute the maximum or the average values of the patch though more complex approaches have been discussed in the literature, e.g. *L2 Pooling* computes the norm of the patch. As for convolutional layers, pooling layers are controlled by the *stride* and *padding* hyper-parameters to reduce or enlarge the size of the output maps.

2.3.3 Inception network

The architecture proposed by [18, 19] addresses some inefficiencies found in the design of deep neural networks, as most architecture are developed by stacking multiple layers of convolutional and pooling layers. Large convolutional kernels capture information from a large spatial region, although they entail high computational costs and can be inefficient when the information content of the region is sparse. Rather than processing large regions, a better approach would be to use smaller kernels that try to find highly correlated patterns between the features, as suggested by the work of [20]. The authors of [18] introduce the *Inception module*, which calculates the convolution operation using several kernels of varying sizes, each aimed at extracting correlation patterns at different scales. The resulting filters are then concatenated to form the final output of the layer. Furthermore, to reduce the costs associated with convolution, the Inception module reduces the input dimensionality by using a 1x1 convolution layer before applying the heavier operations.

Chapter 3

Egocentric action recognition

This chapter introduces the action recognition task and provides an overview of the main contributions from the existing literature. Section 3.1 outlines the objectives of video action recognition and defines the concept of multi-modal learning. Section 3.2 focuses on first-person action recognition, analyzing the peculiarities and critical aspects of this setting compared to the third-person scenario. Finally, 3.4 describes the main deep learning architectures developed for video action recognition.

3.1 Video action recognition

The goal of action recognition is to classify what happens in a video clip, similar to how image classification understands image content by associating samples and labels. Essentially, an action is described by a short phrase made up of a *verb* and a *noun*, e.g. *wash the spoon* or *take the plate*. An *action* is usually short duration and lasts only a few seconds while an *activity* is a long sequence formed by the succession of multiple actions. Although image classification and action recognition may seem conceptually similar, a number of challenges make the latter much more challenging than the former.

Compared to images, videos introduce an element of dynamism, individual frames provide a limited view of what is happening, and predictions must consider information coming from multiple frames. However, simply examining multiple frames may not be sufficient, as the relative order of the frames is also relevant, e.g. to disambiguate between *open* and *close* actions, it is necessary to observe the action in the correct temporal direction. Therefore, a good understanding of the video content comes from an effective combination of both spatial and temporal features. Various methods have been studied to achieve this goal, with several proposals attempting to extend established techniques for image-related tasks.

Figure 3.1: Sample actions from the UCF-101 dataset [21] depicting several human activities from a third-person point of view.

An additional problem that profoundly affects all video-related tasks is the computational and memory costs required for video processing, which increase linearly with respect to the number of frames involved and limits the scope of various deep learning-based techniques.

3.1.1 Multi-Modality

Videos are not limited to RGB frames. In fact, they provide multiple sources of data that contribute unique and complementary details to more informed predictions, an approach known as *multi-modal learning*. The availability of several of these modalities in large-scale datasets, e.g., Epic-Kitchens [12] and Ego4d [25], has fostered the development of new techniques that take advantage of the peculiarities of each modality.

Audio Audio data can provide complementary information to purely visual stream and is not limited by the camera field of view. For example, it is possible to detect whether water in a pot is boiling through audio, even if the pot is not framed. On the other side, audio can easily confuse similar objects made with similar materials, such as a *can* and a *pot* [22]. The authors of [22, 23] analyzed the impact of sound of noun and verb classification over the Epic-Kitchens dataset [8]. Audio alone has

poor performance, but it is useful for improving the accuracy of the verb recognition problem when used in a multi-modal context [24]. Usually, fixed size audio segments are first converted to a spectrogram and processed by a CNN backbone as if they were a 2d image [24]. Therefore, learning from audio is a lightweight task compared to other data sources that incur much higher computational costs.

Optical flow Optical flow [25] detects the direction of motion between adjacent frames, i.e. how the position of pixels is mapped from one frame to the following, providing an hint on which areas of the image are involved by the action. Computing the optical flow remains a computationally expensive task [26] although recent studies have addressed the problem using supervised techniques based on CNNs [27, 28] with significant speedups [29]. More recent developments explored unsupervised optical flow estimation [30]. By focusing only on motion, optical flow is less affected by visual characteristics, e.g. color or text, that are a huge obstacle to generalization.

Figure 3.2: An optical flow frame from the Sintel dataset [2].

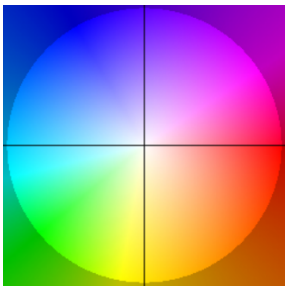


Figure 3.3: The optical flow color coding scheme. Brighter colors indicate smaller motion. Image from [31].

Other modalities Event camera detect sudden brightness changes of individual pixels with a high temporal resolution [32], minimizing the amount of redundant data collected as the background is constantly off. Similarly to optical flow, event stream are quite decoupled with respect to the visual appearance. Unlike optical

flow, events do not incur into high computational costs because they are produced on the fly by special cameras. As such, the authors of [33] explored the possibility of hallucinating optical flow features from event data in order to replace the latter at evaluation time. A similar approach has been explored by [34] to augment RGB data with motion information coming from the optical frames.

Gaze data track the position of the operator's gaze and can be exploited to better identify relevant portions of the field of view [35, 36]. Other modalities include depth and thermal images, textual narrations, 3d scans...

3.2 First person action recognition

Third-person videos capture human activities from an external point of view, as the camera operator does not participate in the actions. In contrast, in egocentric videos, the camera is mounted on the head of the person performing the actions. As a result, the camera naturally follows the ongoing actions thanks to the operator's head movements, providing an occlusion-free view of the objects and interactions involved in the action. Although egocentric viewing suffers from the narrowness of the field of view, which is smaller and in continuous motion and almost chaotic compared to third-person video, it contributes unique features such as gaze direction and hand-object interactions. Both provide natural examples of attention that can be useful in discerning which regions of video are relevant at any given moment, given the key role that hands play in almost every human activity [7].

3.3 Datasets

The application of action recognition techniques to egocentric videos has arisen in recent years along with the availability of large datasets, the main being EPIC-Kitchens [6]. Prior to the release of EPIC-Kitchens, egocentric video datasets were few and limited. Some relevant examples include Charades-ego [37] and EGTEA-Gaze+ [38] though they limited as they include scripted activities and the latter is limited to cooking activities. For its dramatic impact, EPIC-Kitchens can be considered the analogue of Image-Net in the field of egocentric vision.

3.3.1 EPIC-Kitchens

EPIC-Kitchens [6] is a dataset featuring over 55 hours of egocentric videos, collected by 32 participants for a total of 11.5M frames and 39.6K action segments. All the recorded activities are everyday tasks, e.g. brewing coffee or washing the dishes, the participants are highly familiar with. Actions are not scripted and were recorded in the native kitchen environments of the camera wearer, to improve the naturalness of the interactions. Actions are not scripted and were performed for long consecutive

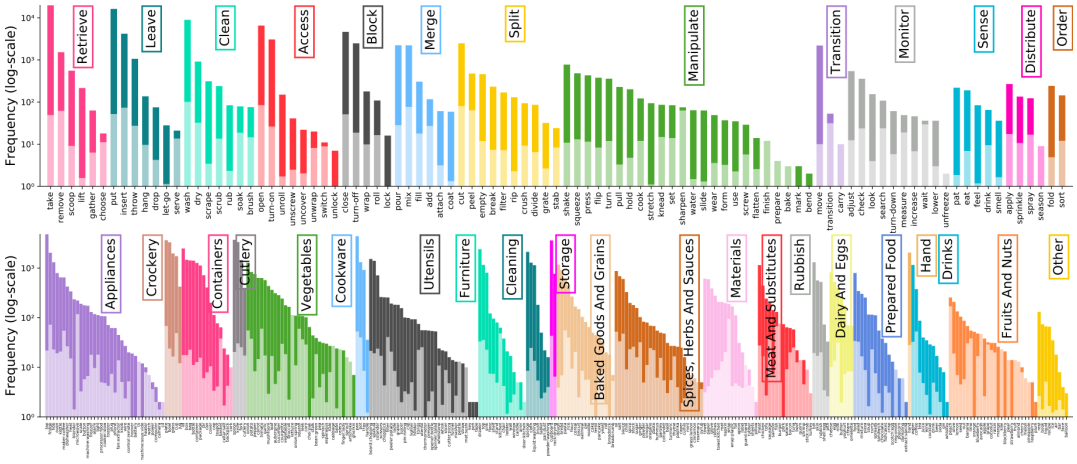


Figure 3.4: Actions and objects distributions in Epic-Kitchens-100 [8].

periods, as participants were asked to keep the camera on for the entire time they spent in the kitchen over a period of three consecutive days. Differences in the nationalities of the participants and in the visual appearance of the kitchens resulted in a very diverse and rich dataset in terms of habits and manners depicted.

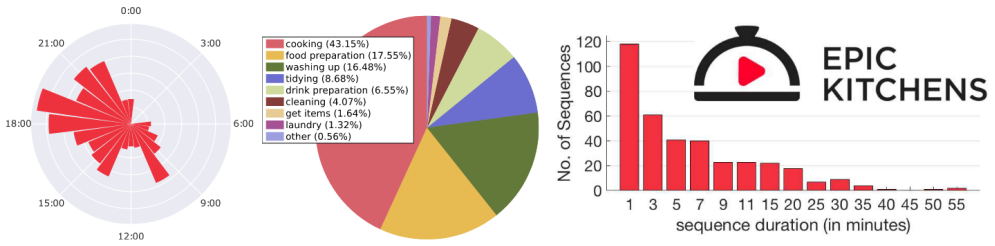


Figure 3.5: Distribution of Epic-Kitchens' samples. Left to right: time of day of the recordings, distribution of actions according to a limited set of macro-categories, duration of the recorded sequences [6].

Videos are captured using a head-mounted GoPro at 1080p resolution and 59.94 *fps* and they are annotated using a combination of coarse annotations provided by the participants and manual transcriptions via Amazon Mechanical Turk. Annotations are not limited to a mere description of the action carried out in each segment but they provide also bounding boxes and time boundaries. For this reason, Epic-Kitchens is fertile ground for other tasks such as action anticipation and object detection.

A recent extension of the dataset, EPIC-Kitchens-100 [8], brought the number of recorded hours to 100 with more than 20 millions frames.

Figure 3.6: Samples from EPIC-Kitchens.

3.4 Architectures

This section describes the main architectures used in video action recognition, from the initial 2D model inspired by image-related tasks to the more recent Inflated 3D ConvNets.

3.4.1 2D ConvNets

Videos are essentially an ordered sequence of frames, which suggests that 2D ConvNet architectures can also be applied to video-related tasks without significant changes. 2D ConvNets have demonstrated to be very effective in the extraction of interpretable features from images although they completely ignore the temporal structure. Early approaches to video classification focuses on extending CNNs with temporal connectivity to learn spatio-temporal features. The authors of [39] proposed an extensive analysis of several techniques to merge spatial and temporal information at varying depth levels in the CNN architecture.

Fusion of spatial and temporal features A first approach consists in computing a prediction for each frame separately, completely ignoring any temporal information and focusing only on the static content of the frame, with remarkable results as discussed by the authors of [39], hinting that sometimes static appearance

may provide enough context information to correctly predict an action.[1] A similar approach is to extract features separately from multiple frames and merge them before computing the final prediction. This approach is referred to as *late fusion*. Fusion can be implemented using different aggregation functions, from concatenation or summation to more complex strategies for re-weighting the contributions of each frame. A conceptually similar technique, *early fusion*, fuses the input frames by extending the first convolutional layer of the network into the temporal dimension, an idea common to many of the following approaches. A middle ground is provided by *slow fusion*, which distributes the fusion operation over all the convolutional layers, progressively mixing spatial and temporal information.

LSTM Although fusion methods can capture motion between different frames, they are commutative and thus fail to recover the relative temporal order of features. Authors of [41] propose to process the features extracted by a 2D ConvNet from each frame using an LSTM layer that models the temporal evolution of features from one frame to its subsequent. The network can produce a prediction at each frame or at the last frame of the sequence.

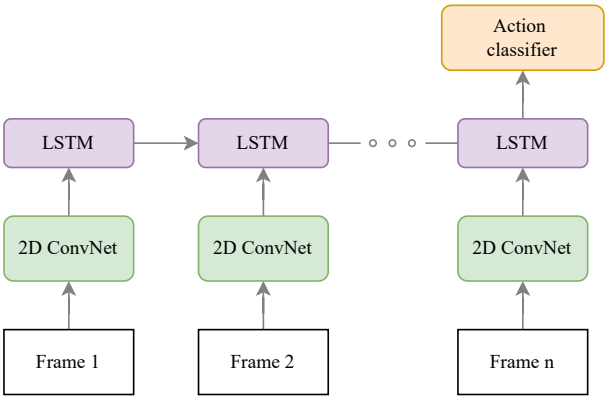


Figure 3.7: 2D-ConvNet + LSTM. The ConvNet component extract spatial features from the frames that are temporally aggregated by the LSTM layer.

[1]The models proposed by [39] derive from the Alex Net architecture [40] and were tested on the Sport-1M dataset [39], a large collection of over a million videos portraying various sport activities. Most labels encode the name of the sport activity, giving rise to a classification task that can often be performed simply by looking at static features of individual frames, such as the environment or sports clothes worn by people.

3.4.2 3D ConvNets

3D ConvNets share the same architecture of their 2D counterparts, but with spatial convolution filters being replaced by spatio-temporal ones [42, 43, 44, 45]. Each clip is fed to the network as a 4d volume with shape $c \times n \times h \times w$ where c is the number of channels, n is the number of frames, h and w are the height and width of each frame. Differently than 2D ConvNets, 3D filters extends in the temporal dimension, enabling joint tracking of both motion and appearance. Experiments

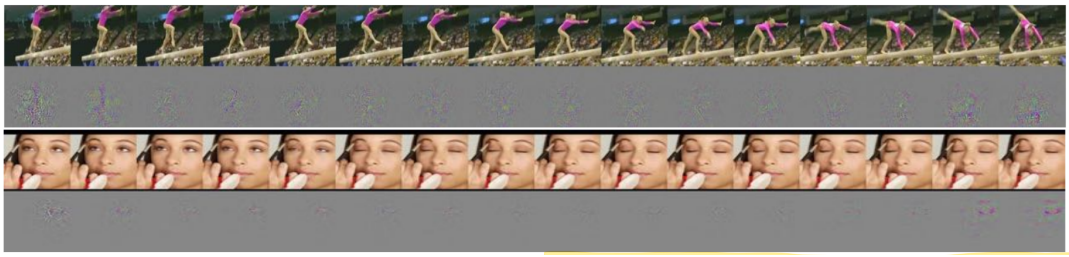


Figure 3.8: Activation maps of C3D [44]. The activation maps of the convolution blocks concentrate on the visual content from the first frames to then focus on smaller details encoding motion.

show that limited spatio-temporal receptive fields are sufficient to achieve good performance. The C3D architecture [44] uses $3 \times 3 \times 3$ kernels.

3.4.3 Multi-Stream Inflated 3D ConvNets

A milestone in the development of deep learning has been the ability to transfer knowledge learned on large datasets, such as ImageNet [46], to other tasks without training from scratch. Indeed, the large number of parameters in 3D ConvNets is an obstacle to large-scale training and limits the otherwise promising performance of this type of network.

The authors of [3] propose to inflate 2D models into 3D ConvNets, by simply adding an extra dimension to all the convolutional and pooling layers, e.g. from $n \times n$ to $n \times n \times n$. The result is an architecture known as *I3D* that can be bootstrapped using the same parameters of the original 2D model, by replicating the weights over the time dimension and rescaling them according to the temporal depth. Given that convolution is a linear operator, the rescaling factor ensures that a video consisting of a single repeated frame, a *so-called* boring-video, has the same pooled activation maps of the original frame.

The I3D architecture can process not only RGB frames but also optical flow inputs to integrate motion information. The authors of [3] propose a two-stream architecture with a late fusion strategy to mix the predictions of the two modalities. This approach, called *multi-modal* learning, is particularly suited for egocentric vision where some modalities, e.g. RGB, are deeply and negatively impacted by

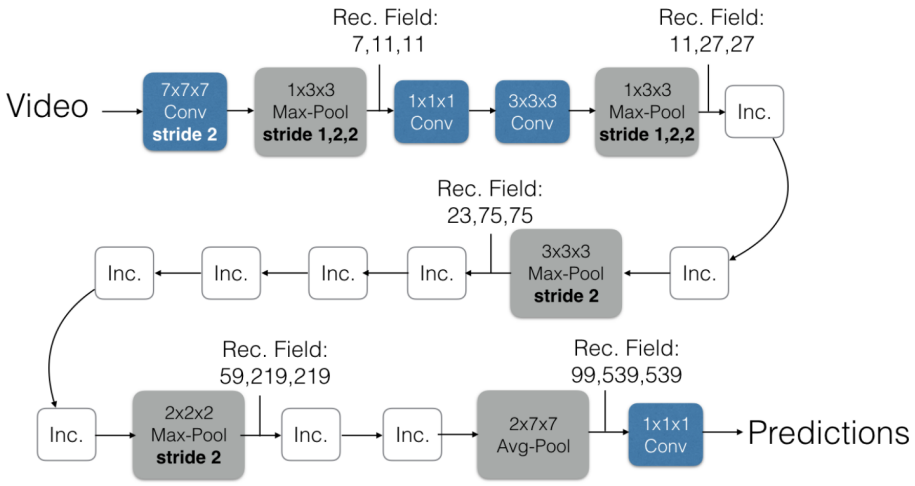


Figure 3.9: The Inflated Inception-V1 architecture [3]. Note that temporal stride of the first two pooling operations is 1. The absence of temporal pooling in the initial layers helps the network preserve spatial details.

some of the peculiar characteristics of egocentric videos, such as egomotion, while others, e.g. Audio, are less affected. A multi-modal network can benefit from the insights brought by each modality to compute more informed predictions.

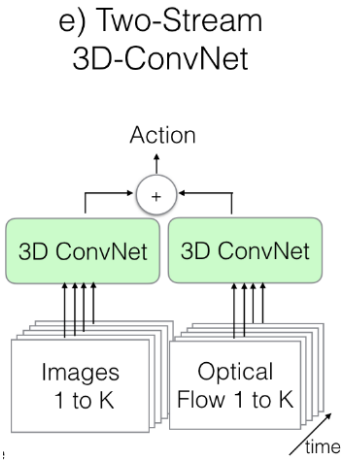


Figure 3.10: Two-stream Inflated 3D ConvNet [3]. RGB and optical flow frames are processed by two different 3D ConvNets to extract both spatial and temporal information. The predictions are then averaged across the two models.

Chapter 4

Domain Adaptation

This chapter provides an overview of different deep learning techniques that seek to transfer knowledge across different domains. Section 4.1 formally introduces the field of Transfer Learning (TL) and presents a classification of the main approaches available in the literature. Section 4.2 focuses on Unsupervised Domain Adaptation (UDA), a particular DA setting in which unlabelled samples from the target domain are available at training time. Finally, Section 4.3 explores the main approaches for UDA in video action recognition.

4.1 Transfer Learning

An important contribution to the success of deep learning comes from its data-driven approach as models learn how to solve tasks directly from data by training on large datasets over long periods of time. However, the strict dependence of the training process on data may prevent the model from learning hypothesis that are *general enough* to be transferred on different data domains. Indeed, it is difficult for models to distinguish between dataset properties that are relevant for the task and others that are simply an insignificant source of bias [47, 48, 49].

The main source of bias is the selection process used to create the dataset, as the samples collected for each class may provide an unrepresentative view of that class. As an example, authors of [49] compare samples of the *car* class belonging to different datasets showing that some datasets only include a limited set different *car* models, e.g. *sport cars*, or have a limited variability in terms of orientation of the image subject, lighting conditions or visual appearance. Deep learning models tend to embed these sources of bias in their decision process, leading to degraded performance when tested with samples having a different visual appearance with respect to the training data. The discrepancy between train and test data is known as *domain shift* and represents a dramatic barrier to the transfer of knowledge across different domains.

Transfer Learning (TL) [50] is a broad field that encompasses all techniques oriented toward the transfer of hypotheses learned over one or more source domains to a labeled or partially labeled target domain.

4.1.1 Formal introduction

Following the notations proposed by [51, 52, 53], a domain $\mathcal{D} = \{\mathcal{X}, \mathcal{T}\}$ is defined by a data space $\mathcal{X} = \{x_i\}$ distributed according to $P(\mathcal{X})$ and by a task $\mathcal{T} = \{\mathcal{Y}, P(\mathcal{Y}|\mathcal{X})\}$ that defines a conditional distribution $P(\mathcal{Y}|\mathcal{X})$ between $\mathcal{X}$ and a label space $\mathcal{Y}$. In a typical machine learning problem, a model $\mathcal{M}$ is trained on a source domain $\mathcal{D}^s = \{\mathcal{X}^s, \mathcal{T}^s\}$ to learn a predictive function h that is then evaluated on the target domain $\mathcal{D}^t = \{\mathcal{X}^t, \mathcal{T}^t\}$. In the simplest setting the source and target domains coincide, i.e. $\mathcal{D}^s = \mathcal{D}^t$ and $\mathcal{T}^s = \mathcal{T}^t$, and the predictive function h can be directly applied to the target domain without any additional tuning. The scenario in which both the training and the evaluation samples belong to the same dataset falls under this case. On the other side, if the source and target domains are different, i.e. $\mathcal{D}^s \neq \mathcal{D}^t$, the h function learned on the source domain is not valid anymore on the target. The objective of TL is to favour the transfer of knowledge learned on one domain to a different one.

Homogeneous and heterogeneous TL Depending on the similarity between source and target data, the authors of [53] distinguish two variants of TL. In *homogeneous TL*, source and target share the same representations $\mathcal{X}^s = \mathcal{X}^t$ even though the two marginal probability distributions may differ, i.e. $P(\mathcal{X}^s) \neq P(\mathcal{X}^t)$. For example, in a classification problem, both $\mathcal{X}$ and $\mathcal{Y}$ can represent images, but belonging to different datasets and with different distributions.

In *heterogeneous TL*, source and target have different representations and therefore also different distributions. This scenario is typical of multimodal problems as different modalities can be used to represent the same information.

Inductive and transductive TL The goal of *Inductive TL* is to address a task on a partially labeled or unlabeled target domain by learning *meta*-knowledge that is not directly related to the task [54, 50]. Learning additional concepts that are not directly related to the target task may be useful to indirectly *induce* improvements in the main objective, possibly exploiting the labels of the source domain if available. For example, in image classification models, invariance with respect to affine transformations of the input image is a useful precondition for learning more robust hypotheses. Another common approach consists in projecting source and target data in a shared low-dimensional subspace where the differences between the two domains are less noticeable.

In *transductive TL*, the task is shared between the two domains even though the two datasets differ in their representation, i.e. $\mathcal{X}^s \neq \mathcal{X}^t$, or probability distribution,

$P(\mathcal{X}^s) \neq P(\mathcal{X}^t)$.

Unsupervised TL is the most difficult setting, since both tasks and datasets are different though related, and no label is provided.

4.2 Unsupervised Domain Adaptation

Unsupervised Domain Adaptation (UDA) is a particular *transductive TL* setting where the source and target share the same task $\mathcal{T}^s = \mathcal{T}^t$ and the same representations $\mathcal{X}^s = \mathcal{X}^t$ but have different probability distributions $P(\mathcal{X}^s) \neq P(\mathcal{X}^t)$. This chapter focuses on common deep learning methods for UDA that learn to bridge the domain gap directly from raw data. Loosely following the classification proposed by [52], UDA methods can be organized into the following categories, depending on the approach used.

- *Discrepancy-based*: these methods attempt to reduce the discrepancy between the generated features across different domains by minimizing a distance metric between the two [55, 56].

- *Adversarial networks*: methods in this category typically add a domain discriminator that distinguishes between source features and target features. The goal of the feature extractor is to increase classifier confusion, and to do this it must produce features that are more difficult for the classifier to distinguish, that is, more invariant between different domains [57, 58].

- *Pseudo-labelling methods*: the model pretrained on the source domain may be used to predict pseudo-label for the target domain, to overcome its lack of labeled samples.

- *Reconstruction-based methods*: these methods learn to decouple *domain-specific* and *domain-invariant* features with the use of autoencoders pairs. Among these methods, Domain Separation Networks (DSN) [59] introduce a shared autoencoder to learn *domain-invariant* representation and a separate autoencoder for each domain to learn *domain-private* representations. Representations learned from the shared network are encouraged to be similar across domains, while private autoencoders learn dissimilar representations to encode the domain specific information.

4.2.1 Discrepancy-based UDA

Discrepancy-based methods promote the alignment of features across different domains.

MK-MMD A naive approach to the comparison of two probability distributions would be to look at their empirical means even though the mean is usually not enough to properly describe a probability distribution. A more general approach is provided by Maximum Mean Discrepancy (MMD) [55] which allows to compare probability distributions in an higher and possibly infinite dimensional space. With MMD, the distance between two distributions P and Q over $\mathcal{X}$ is computed in an Hilbert space $\mathcal{H}_k$, defined by a mapping $\phi : \mathcal{X} \to H_k$ and by a kernel $k(x, y) = \langle \phi(x), \phi(y) \rangle$ that allows to compute the inner product in the high-dimensional space without an explicit mapping. If the mapping is the identity function, then the resulting MMD formulation is equivalent to the distance between the means of the two distributions.

$$\begin{aligned} d_k^2(P, Q) &\triangleq ||\mathbb{E}_{x\sim P}[\phi(x)] - \mathbb{E}_{y\sim Q}[\phi(y)]||^2_{\mathcal{H}_k} \\ &= \mathbb{E}_{x,x'\sim P}[k(x, x')] + \mathbb{E}_{y,y'\sim P}[k(y, y')] + \mathbb{E}_{x\sim P,y\sim P}[k(x, y)] \end{aligned} \tag{4.1}$$

The Deep Adaptation Network proposed by [60] adds a regularization term to the CNN loss to reduce the discrepancy between the high-level features learned from source and target samples at the top l layers.

$$\mathcal{L} = \mathcal{L}_c(\{x_i^s, y_i^s\}) + \lambda \sum_l d_k^2(\mathcal{D}_l^s, \mathcal{D}_l^t) \tag{4.2}$$

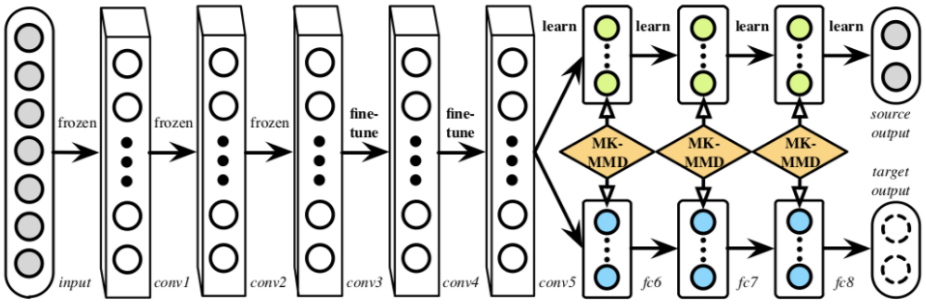


Figure 4.1: Deep Adaptation Network (DAN) proposed by [60] and adapted from AlexNet [61]. The first layers are either frozen (*conv1-conv3*) or fine-tuned (*conv3-conv4*). The MK-MMD loss is applied to the features extracted by the top fully connected layers as they are closer to the specific task.

CORAL *CORAL* [56] aligns the second order moment of the high-level features extracted across different domains and acts as a regularization term alongside a traditional supervised loss. Formally, the CORAL loss is defined as the Frobenius norm of the difference between the covariance matrices, C_S and C_T, of the source and target features extracted by the last fully connected layer of the network.

$$\mathcal{L}_{CORAL} = \frac{1}{4d^2}||C_S - C_T||_F^2 \tag{4.3}$$

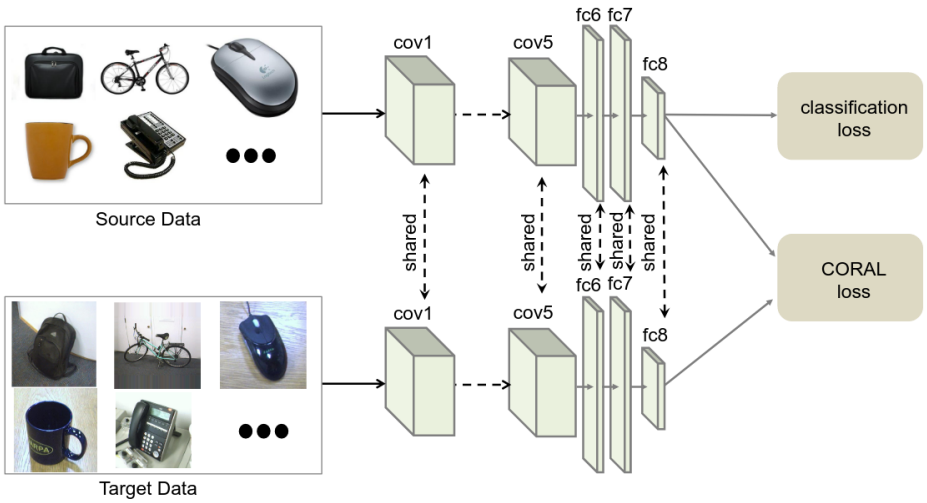


Figure 4.2: DeepCoral Architecture [56]

Other discrepancy-based approaches Kullback-Liebler divergence (KL) [62] and Jensen-Shannon Divergence (JSD) [63] measures the divergence between two probability distributions and may be used to align the predictions produced by a model across different domains [64, 65]. Other approaches exploit Batch Normalization layers to mix normalization statistics from different domains [66] or to use domain-specific normalization [67].

4.2.2 Adversarial methods

Adversarial methods for domain adaptation aim at extracting domain invariant features by optimizing the supervised task on the source domain and a domain discrimination task on source and target data. The adjective *adversarial* suggests that the two tasks are treated as opposite objectives to be minimized at the same time, as in a game. The concept of multiple networks competing against each other to achieve different goals has became popular in recent years with the advent of Generative Adversarial Networks (GAN) [68].

Domain Adversarial Neural Network

The adversarial architecture proposed by [57] is composed of a shared backbone for features extraction and two classifiers, one that predicts class labels on source samples (*label predictor*) and a second that outputs a domain label (*domain classifier*), e.g. *source* or *target*. Overall, the network objective is defined as a weighted sum of two supervised contributions: a cross-entropy loss $\mathcal{L}_c$ for the label predictor and

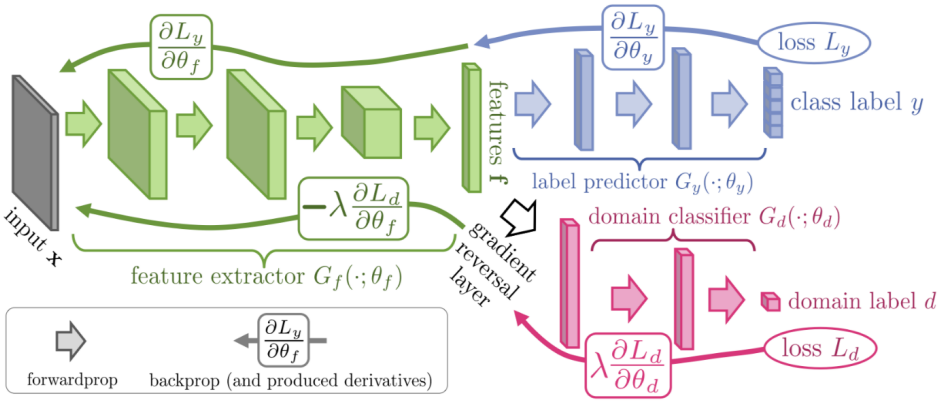


Figure 4.3: Domain Adversarial Neural Networks [57]. The gradient of the classification loss $\mathcal{L}_c$ flows end to end with modifications. On the contrary, the gradient of the domain classification loss $\mathcal{L}_d$ is propagated unchanged through the domain classifier layer and reversed in sign when it passes through the feature extractor.

a binary cross-entropy loss $\mathcal{L}_d$ for the domain classifier.

$$\mathcal{L} = \mathcal{L}_c(\mathcal{D}^s) + \lambda\mathcal{L}_d(\mathcal{D}^s, \mathcal{T}^s) \tag{4.4}$$

Without any additional constraint, the optimization of the loss in Eq. 4.4 would result in intermediate features that are highly discriminating as a result of the minimization of the domain classification loss. Ideally, the domain classifier should be as good as possible at distinguishing between source and target samples, while the feature extractor should focus on producing features that are not so easy to classify. To enforce this constraint, the author of [57] introduce a *Gradient Reversal Layer (GRL)* between the features extractor and the domain classifier. The GRL implements a pseudo function $R_\lambda(\mathbf{x})$ that behaves like an identity function in the forward pass and that has a negative gradient in the backward pass.

$$R_\lambda(\mathbf{x}) = \mathbf{x} \qquad \nabla_{\mathbf{x}} R_\lambda(\mathbf{x}) = -\lambda\mathbf{I} \tag{4.5}$$

This trick effectively forces the feature extractor to learn more difficult features to classify as it receives gradient updates that maximize the classifier's loss. Indeed, the feature extractor and the domain classifier are adversaries as they pursue opposite goals.

Adversarial Discriminative Domain Adaptation

The *Adversarial Discriminative Domain Adaptation (ADDA)* strategy proposed by [58] is conceptually similar to DANN. Starting with a network pretrained on the source domain, ADDA seeks to train an encoder on the target domain to obtain

intermediate features as indistinguishable as possible from those of the source domain. Then, the resulting encoder is used to replace the feature extractor of the pretrained network to classify the target samples. A noticeable advantage of ADDA compared to DANN is that it does not require access to source data to bridge the domain gap.

4.3 UDA for Action Recognition

Compared to more general scenarios, UDA approaches for action recognition can potentially exploit the unique characteristics of videos, namely their temporal dimension and multi-modal nature. Although the number of UDA methods specifically designed for action recognition is limited, this section describes the most important approaches in this field.

4.3.1 MM-SADA

The *Multi-Modal Self-Supervised Adversarial Domain Adaptation (MM-SADA)* architecture proposed by [69] is one of the first attempts at exploiting the multi-modal nature of videos for domain adaptation. MM-SADA extends the approach of DANN to learn features with a low discrepancy across different domains and different modalities. It incorporates M feature extractors F^m, one for each modality m, which are encouraged to learn features that are similar across different domains through the *adversarial* loss L_d^m computed by classifier D^m, as previously discussed in [57]. Given a sample x belonging to domain d, the corresponding L_d^m loss for modality m is given by

$$\mathcal{L}_d^m = \sum_{x \in \{S,T\}} -d \log \left(D^m(F^m(x))\right) - (1-d) \log \left(1 - D^m(F^m(x))\right) \tag{4.6}$$

To facilitate the alignment of features across different modalities, MM-SADA introduces an additional *self-supervised* binary classification task through the classifier C which is asked to distinguish between features $(F^0(x), \ldots, F^M(x))$ sampled from the same or different actions. A binary label c identifies whether or not all modes correspond to the same action.

$$\mathcal{L}_c = \sum_{x \in \{S,T\}} -c \log C(F^0(x), \ldots, F^M(x)) \tag{4.7}$$

Finally, MM-SADA adopts a late fusion strategy as features from different modalities are fed to separate classifier whose outputs are then averaged to compute the final predictions. Overall, the training loss $\mathcal{L}$ of MM-SADA is a sum of three contributions to encourage alignment across different domains $\mathcal{L}_d$ and different modalities $\mathcal{L}_c$ while solving the main supervised task $\mathcal{L}_y$ using the standard cross-entropy loss.

$$\mathcal{L} = \mathcal{L}_y + \lambda_c \mathcal{L}_c + \lambda_d \sum_m \lambda_d^m \tag{4.8}$$

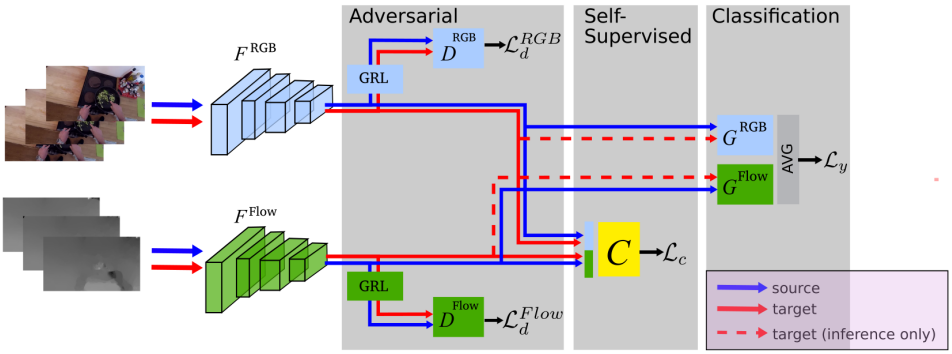


Figure 4.4: Multi-Modal Self-Supervised Domain Adaptation [69]. Source (blue) and target (red) samples are fed to two features extractor F^{RGB} and F^{Flow}. Two classifiers D^{RGB} and D^{Flow} learn to distinguish between source and target features by computing a classification loss that is backpropagated through the feature extractors via a Gradient Reversal Layer. The classifier C discriminates features according to whether or not they were drawn from the same action. Finally features are sent to two classifier, G^{RGB} and G^{Flow}, and the outputs are averaged to compute the final predictions and the corresponding supervised classification loss $\mathcal{L}_y$.

4.3.2 CoMix

One could argue that an approach such as MM-SADA does not emphasize the temporal dynamics of videos. Indeed, the self-supervised loss $\mathcal{L}_c$ of MM-SADA shows similar benefits on the overall performance, regardless of whether the samples come from the same time instant of the action(s) or from more distant segments, suggesting that the proposed architecture is not giving much importance to the temporal dimension of the samples, possibly focusing more on static and time-invariant features [69]. To address this issue, CoMix [70] *maximizes the similarity* between multiple representations of the same video encoded at different speeds and *minimizes the similarity* between the representations of different videos. To bridge the domain gap between different domains CoMix proposes to mix frames from one domain with the background from another domain. The result retains the *motion pattern* of the original frames and allows to define, for each clip, multiple positive anchors combining slow and fast versions of the same clip as well as mixed background.

Moreover, CoMix assigns pseudo-labels to target samples whose predictions are above a certain confidence threshold and moves closer in the embedding space samples with the same label using a contrastive loss. Finally, a standard cross-entropy loss is minimized over the labeled samples of the source dataset.

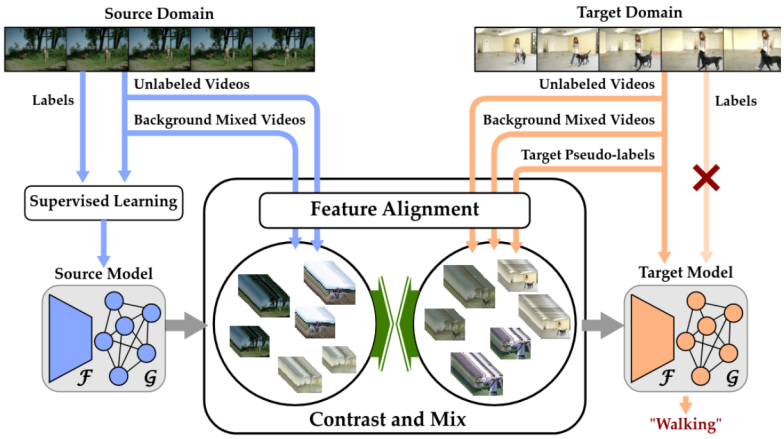


Figure 4.5: CoMix architecture [70]. CoMix aligns features using multiple positive anchors: a) fast and slow versions of the same video, b) samples with mixed background and c) samples from the target domain with same pseudo-label.

4.3.3 TA3N

TA3N extracts features from each frame using a ResNet backbone [71] pretrained on ImageNet [72]. To learn features that are more task-specific, the output of the backbone is further processed by multi-layer perceptron (MLP) G_{sf}. As in [57, 69], TA3N exploits an adversarial domain discriminator G_{sd} to learn spatial features with a smaller gap across different domains. To aggregate features on the temporal axis, TA3N follows the approach of [73] to fuse information from multiple temporal-ordered set of frames at different time scales.

4.3.4 RNA

The authors of [1] observed that training multi-modal models result in an *unbalance* between the different modalities, as one tends to prevail over the others by having a larger norm even if its information content is not necessarily larger. Experimental results show that when RGB and audio models are trained together on Epic-Kitchens, the audio features have an L2 norm almost three times larger than that of RGB. To solve this problem, the proposed RNA loss forces the network to learn how to rebalance the contributions of the different modalities by minimising the distance between their average feature norms.

$$\ell_{RNA} = \left(\frac{\mathbb{E}[h(X^v)]}{\mathbb{E}[h(X^a)]} - 1 \right)^2 \tag{4.9}$$

where $h(\cdot)$ denotes the L2-norm while X^v and X^a represents the features extracted from the RGB and Audio modalities respectively.

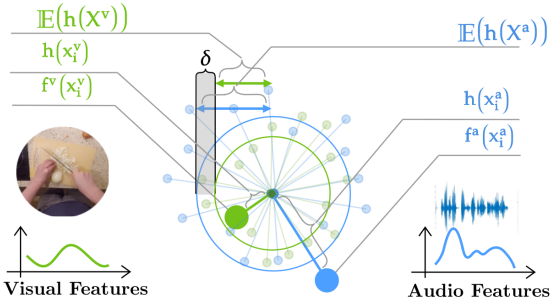


Figure 4.6: Features norm-unbalance of the RGB and Audio modalities. The green and blue dots represents the mean RGB and Audio features respectively and δ measures the distance between the norm of the two. Minimization of the RNA loss reduces the distance δ, encouraging the mean features to have the same norm.

Visually, the average features of the two modalities are on two hyper-spheres of different radius and the goal of RNA is to make the radius equal for both modalities. In the process, the angle between the feature is not affected, unlike other loss functions such as cosine similarity that would make the average features more parallel, leaving more freedom to the model to select the most suitable alignment. RNA proves to be a viable loss function for both DG and UDA settings, as it can

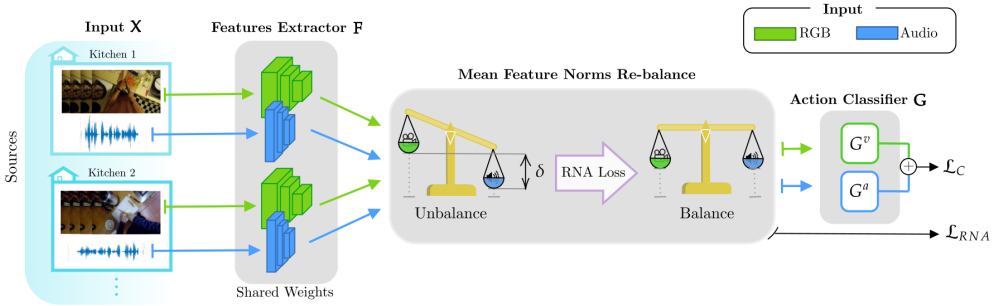


Figure 4.7: RNA-Net [1]. The goal of the RNA loss $\mathcal{L}_{RNA}$ is to align the mean features norms of the two modalities computed over possibly multiple source domains.

improve the alignment of norms between different domains, allowing the network to learn equally from each of them.

Relative Norm Alignment by Class (RNA-C)

RNA pushes the network to learn mean features that have the same norm across different modalities, enabling the models to learn equally from each of them. However, this approach does not consider the informative content that comes with each modality. Indeed, some actions are more easily recognisable by looking at RGB frames, while others rely more on audio clips, suggesting that the balance between

the different modalities depends very much on the type of action involved. Based on this assumption, RNA-C extends the RNA loss by allowing the network to freely choose the best norm value for each class of actions. The usual RNA loss is computed on the samples of each class c separately and the contributions are summed up according to the population of each class N_c.

$$\ell_{RNA-C} = \sum_{c=1}^{K} \frac{1}{N_c} \left(\frac{\mathbb{E}[h(X_c^v)]}{\mathbb{E}[h(X_c^a)]} - 1 \right)^2 \tag{4.10}$$

Furthermore, in a DG or UDA setting, RNA-C encourages the network to learn features that have the same norm when extracted from the same action, even if they come from different datasets, possibly easing the generalisation process. Unlike RNA, C-RNA is a supervised loss in that it requires the knowledge of class labels to group features according to the action from which they were extracted. To overcome this problem, a common self-supervised approach, known as *pseudo labelling*, uses the network predictions instead of the ground truth labels, under the optimistic assumption that the class assignments produced by the network are correct.

Chapter 5

Test Time Adaptation

This chapter formally introduces Test Time Adaptation (TTA), with a focus on its differences from the common approaches of Domain Adaptation and Domain Generalization. Section 5.2 focuses on one of the most trivial but powerful methods to address the domain shift problem, namely the update of the Batch Normalization statistics. Finally, the sections 5.3 and 5.4 provide an overview of some of the most common loss functions used in TTA.

5.1 Introduction

Traditional techniques for domain adaptation and domain generalization are difficult to extend to the real-world scenarios. The goal of domain generalization is to learn more robust predictive functions, usually exploiting multiple source domains that share the same label space, as an attempt to lower the prediction error on an unseen target domain. The application of domain generalization techniques requires us to consider certain assumptions. First of all, the number of available source domains to learn from is usually limited and extrapolating formal guarantees on the average risk estimation error bound is not a straightforward process, as addressed by [74]. Indeed, even with a substantial number of source domains, the gap between their data distribution and the distribution of target samples may still be too large, leading to little or no improvement over similar models trained on fewer domains. Moreover, literature on domain generalization applications to actions recognition is rather limited and mostly focused on image-related tasks. Authors of [75] propose a DG architecture, specifically designed for videos showing significant improvements with respect to other approaches originally developed for image tasks. Similarly, recent work by [1] addresses domain generalization by exploiting the multimodal nature of videos to prevent one modality from prevailing over the others.

Domain generalization approaches do not exploit the knowledge provided by

the target domain samples, even if unlabeled. The goal of Unsupervised Domain Adaptation (UDA) [51], on the other hand, is to improve performance on a target dataset by learning from the target samples in an unsupervised way, making it less realistic than DG but more effective in practical applications. As already seen for DG, UDA also requires certain assumptions, mainly related to the availability of target data at the time of training. Models must be trained from scratch on each new pair of source and target domains $(\mathcal{D}^s, \mathcal{D}^t)$. Training usually takes time and energy, which prevents the implementation of UDA techniques on end devices with limited computational capabilities, such as the lightweight and inexpensive wearable cameras used in egocentric vision. For this reason, training is usually performed on powerful workstations on which both source and target data reside. Moving videos outside the device that generated them is a possible privacy violation, as well as costly in terms of time and power required for transmission. Finally, the most critical issue is the existence and uniqueness of a target domain. Often the target domain may be not only unavailable but even nonexistent at training time as the scope of application of the model is not known in advance or it is dynamically changing.

An alternative approach, *Source-Free Domain Adaptation (SFDA)*, relies only on unlabeled target data to perform multiple *iterative* training steps, usually exploiting an auxiliary task, as in [76]. However, the problem of target data availability still remains.

Test-Time Adaptation (TTA) solves these problems by shifting the burden of adapting models to unknown domains to the testing phase, on the assumption that the available samples provide indications of the true distribution of the target domain. More formally, the goal of TTA is to update the hypothesis h learned from possibly multiple labelled source domains $\mathcal{S}_k = \{(x^s_{k,i},\ y^s_{k,i})\}_i$ to the current target domain $\mathcal{T} = \{(x^t_i)\}_i$ from which the model is sampling.

$$h : \mathcal{X}_\mathcal{S} \to \mathcal{Y}_\mathcal{S} \qquad \xrightarrow{TTA} \qquad \tilde{h} : \mathcal{X}_\mathcal{T} \to \mathcal{Y}_\mathcal{T} \tag{5.1}$$

As labels are not available for the target domain, it is assumed that the two label spaces, $\mathcal{Y}_\mathcal{S}$ and $\mathcal{Y}_\mathcal{T}$, coincide. In addition, the absence of target labels limits the number of techniques that can be applied to improve the quality of the model assumptions. Typical approaches for TTA can be classified into two main classes, depending on whether or not they require gradient updates through backpropagation.

Backpropagation-free TTA A first class of TTA approaches focuses on updating the Batch Normalization statistics collected by the model at training time. These methods do not update model parameters and therefore do not require backpropagation. Replacing normalization statistics collected on the training data with online estimates of the target data has been shown to improve the robustness of the model in presence of covariance shift [77]. More details in Section 5.2.

Loss-based TTA The goal of class-related losses is to improve the quality of predictions, such as improving the strength of the outputs by optimizing some side properties of the class distribution produced by the model, like its entropy. This is achieved by minimizing a loss function calculated on the model results. Several class-relative losses are analyzed in Section 5.3. Similarly, feature level losses attempts to improve the model outputs at the feature level, e.g. before computing the logits. More details in Section 5.4.

	Training phase		Testing phase	
	Source data	Target data	Target data	Online
UDA	✓	✓	-	✓
DG	✓	-	-	✓
SFDA	-	-	✓	✗
TTA	-	-	✓	✓

Table 5.1: Summary of several Domain Adaptation techniques. UDA and DG do not train the model at testing time. SFDA performs iterative training on target data, making it not suitable for online applications. TTA trains the model online on incoming target data.

5.2 Batch normalization

The first class of methods for TTA derives from the assumption that domain shift primarily results in a deviation from the batch normalization statistics collected at training time. A trivial example is represented by the batch normalization layers close to the input. Changes in the environment, e.g. furniture colour or light conditions, may drastically modify the mean and variance of the frame's channels observed by the normalization layers.

A natural solution is to replace the running statistics of the normalization layers with the mean and variance of the target samples [77, 78]. Prediction-time normalization ensures that the output of the normalization layers remains in the same range encountered at training time, as this is the only portion of the input space explored during training. Outside this region, model behavior can become unpredictable. The authors of [77] also suggests that prediction-time normalization prevents the layers from projecting out of distribution samples into regions that may result in highly confident predictions.

Depending on the magnitude of the shift between the source and target distributions, the replacement of normalization layer statistics may result in a significant discrepancy between the activations seen by the model at the time of training and testing. In other words, prediction-time normalization effectively reduces the domain shift problem, possibly at the cost of increased model uncertainty. To alleviate

this problem, the authors of [79] propose to mix the source and target statistics. The approach, called α-BN, computes, for each BN layer i, a new set of normalization statistics $\{\tilde{\mu}^{(i)}_{new}, \tilde{\sigma}^{(i)}_{new}\}$ as a weighted average of the old estimates computed at training time over the source data $\{\tilde{\mu}^{(i)}_{src}, \tilde{\sigma}^{(i)}_{src}\}$ and the running estimates over the batch of target samples x_t, effectively mixing hints coming from both domains.

$$\hat{\mu}^{(i)}_{new} = (1-\alpha)\hat{\mu}^{(i)}_{src} + \alpha\mathbb{E}(x_t) \qquad \hat{\sigma}^{(i)}_{new} = (1-\alpha)\hat{\sigma}^{(i)}_{src} + \alpha\sqrt{\text{var}(x_t)} \tag{5.2}$$

$\alpha \in [0, 1]$ is an hyper-parameter controlling the balance between the old and new estimates. If $\alpha = 0$, the statistics are never updated while $\alpha = 1$ completely neglects the old estimates. The formulation is completely equivalent to the update rule employed by BN at training time with α being the *momentum* hyper-parameter. Therefore, the implementation of this adaptation approach simply requires to put the layer in training mode and to select a proper value for α.

Experiments of [79] prove that updating the BN statistics plays a key role in the adaptation process, leading to very positive results. Moreover, α-BN appears to be quite robust to the choice of α with respect to the task [79]. Finally, the method does not require backpropagation and can be easily combined with the other adaptation techniques described in the following sections.

5.3 Class Relative losses

5.3.1 Entropy Minimization (ENT)

In information theory, the *entropy* of a probability distribution measures the uncertainty of its outcomes. If all the outcomes are equally likely, the distribution is said to have a high *information content* or an *high entropy*. If, on the other hand, all outcomes have a very low probability of occurring, with one exception having a much higher probability, then there is not much uncertainty about the outcomes and the distribution has *low entropy*. In a classification task in which each sample is to be assigned to only one class, it is preferable for the model to produce predictions with very low entropy as an higher entropy is a sign of *confusion* in the predictive function learned by the model.

Network predictions usually show higher entropy on the unseen data because of the shift in internal covariance produced by the change in conditions from the time of training, as shown by Table 5.2. The authors of [80] propose the use of entropy in supervised learning as an effective regularizer, provided that the classes are not overlapping. Similarly, a natural solution might be to update the network weights along the direction that minimizes the prediction entropy, in an effort to strengthen the network's confidence.

$$\ell_{ent}(\mathbf{y}) = -\frac{1}{n}\sum_i \sum_c y_{i,c} \log y_{i,c} \tag{5.3}$$

Minimization of prediction entropy does not guarantee the best solution, as the model may become even more radicalized on incorrect predictions. The authors of [81] discuss the possibility of running into degenerate solutions when using entropy minimization alone, as the network may always predict the same class regardless of the input, effectively minimizing entropy loss but failing to solve the main task correctly. Moreover, the formulation of the entropy loss gives all the samples the

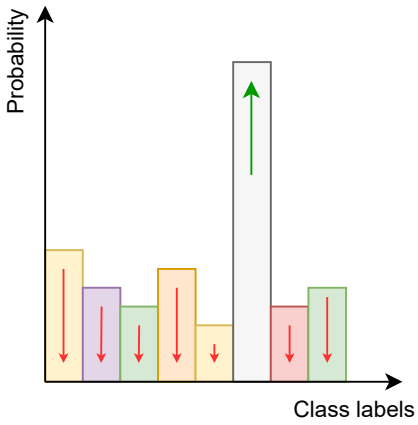


Figure 5.1: Entropy minimization (ENT). The goal of entropy minimization is to create a peak in the distribution of labels, flattening the probability of all remaining labels. The arrows show the effect of the loss on the model predictions.

same weight, possibly biasing the network predictions towards the majority classes of the batch.

	Target domain		
Source domains	D1	D2	D3
D2-D3	**1.2537**	1.0941	1.1389
D3-D1	1.1651	**1.2773**	1.2065
D1-D2	1.0299	0.9738	**1.1346**

Table 5.2: Entropy of predictions on source and target datasets using an *RGB+Flow (RNA)* model without any adaptation technique over the three largest kitchens in Epic-Kitchens [6]. Values in **bold** highlight the increase in entropy when the model is applied to samples from unseen domains. More details on the configuration used for the experiments are provided in Section 6.

5.3.2 Information Maximization (IM)

Information Maximization (IM) [82, 83, 84] mitigates the detrimental effect of the entropy loss in the presence of an unbalanced test dataset by enforcing diversification in the model predictions. The IM loss ℓ_{im} is defined as the sum of two component: the average entropy of the samples, as in the entropy minimization formulation 5.3, and the negative entropy of the average samples. The first component aims to reduce the uncertainty of the predictions, while the second ensures that they remain globally different to avoid the pitfalls of entropy minimization alone.

$$\ell_{im}(\mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log y_{i,c} + \sum_{c=1}^{C}\tilde{p}_c \log \tilde{p}_c \quad \textit{where} \quad \tilde{p}_c = \frac{1}{n}\sum_{i} y_{i,c} \tag{5.4}$$

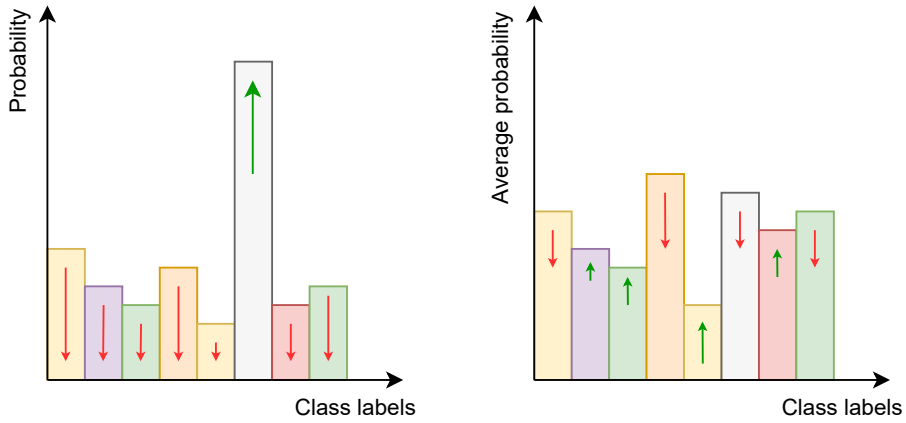


Figure 5.2: Information Maximization (IM). The goal of information maximization is to create a peak in label distribution (*left plot*), like entropy minimization, while ensuring that the model's average predictions remain globally diverse, thus avoiding trivial solutions (*right plot*).

Although IM represents an improvement over entropy minimization, it still suffers from an assumption about the distribution of samples in the batch. In fact, imposing globally diverse predictions may be not optimal if the batch contains a small number of unique classes and may even lead to increase output uncertainty. This problem is of particular concern in the context of TTA, as batches should ideally be as small as possible to allow near real-time processing.

5.3.3 Minimum Class Confusion (MCC)

Unlike entropy and information maximization, the *Minimum Class Confusion (MCC)* loss, proposed by [85], is a versatile domain adaptation approach that targets a reduction of the *pairwise class confusion* of the model predictions, i.e. the situation

in which a sample is ambiguously classified into two different classes class c_i and c_j with an equally high probability. Entropy fails to detect such conditions, as small perturbations of the probability distribution may have significant impact on the entropy.

As proposed by the authors of MCC [85], the degree of pairwise confusion over the model predictions for two classes i and j is measured using a similarity function, e.g. the dot product, of the vectors $y_{\cdot,i}$ and $y_{\cdot,j}$, which represent the probabilities that the samples in the current batch belong to classes i and j, respectively. More compactly, this is equivalent to calculating a confusion matrix defined as

$$\mathcal{C}_{i,j} = c_{\cdot,i}^T c_{\cdot,j} \tag{5.5}$$

Given that the real labels of the samples are not available, $\mathcal{C}$ is an approximation of the real confusion matrix. Ideally, the confusion matrix should be close to the identity matrix, indicating low *between-class* confusion and strong prediction confidence.

The contribution of each sample can be controlled by a weighting mechanism to give more importance to samples with a low entropy. In fact, high entropy is a symptom of model uncertainty and learning from uncertain predictions is not useful as it might simply increase the confusion. Furthermore, favouring samples with a lower entropy allows the loss to focus on predictions for which the network is somewhat confident. Therefore, entropy can be considered as a measure of how much the model can learn from each sample. In the MCC formulation, the entropy values of the samples are transformed into a probability distribution using the softmax function with Laplace Smoothing (Eq. 5.6).[1] W_{ii} is the weight associated to sample i. The B factor in the weights computation scales the contributions so that $\mathbb{E}_i[W_{ii}] = 1$.

$$W_{ii} = \frac{B(1 + \exp(-\mathcal{H}(\mathbf{y}_{i,\cdot})))}{\sum_{i=1}^{B}(1 + \exp(-\mathcal{H}(\mathbf{y}_{i,\cdot})))} \tag{5.6}$$

The W matrix is used to reweight the contribution of the samples to the class confusion matrix (Eq. 5.7).

$$\mathcal{C}_{i,j} = c_{\cdot,i}^T W c_{\cdot,j} \tag{5.7}$$

Depending on the diversity of the current batch, the nonzero values of the confusion matrix $\mathcal{C}$ may be concentrated in smaller regions of the matrix, corresponding to the labels predicted by the network. To rebalance the contributions of the different classes, the MCC normalizes each row of $\tilde{C}$ so that the sum is equal to 1 (Eq. 5.8). In this way, the confusion matrix can also be interpreted in terms of a random walk

[1] Laplace Smoothing better highlights positive samples without over penalizing the negative ones.

matrix, whose entry $\tilde{\mathcal{C}}_{ij}$ defines the probability that a sample classified as belonging to class i would also be classified as belonging to class j.

$$\tilde{\mathcal{C}}_{i,j} = \frac{\tilde{\mathcal{C}}_{i,j}}{\sum_{k}^{C} \tilde{\mathcal{C}}_{i,k}} \tag{5.8}$$

Finally, the MCC loss ℓ_{MCC} is defined as the sum of all off-diagonal values of $\tilde{\mathcal{C}}$. The minimization of the MCC loss directly reduces the *between-class* confusion and indirectly improves the confidence of the model predictions by moving the confusion matrix towards the identity matrix.

$$\ell_{MCC} = \frac{1}{C} \sum_{i=1}^{C} \sum_{j \neq i}^{C} \tilde{\mathcal{C}}_{i,j} \tag{5.9}$$

Although MCC loss was originally proposed for traditional domain adaptation scenarios, to be used as a secondary task on unlabeled target data in combination with a primary task on source data, it can be extended to test-time adaptation without any modification.

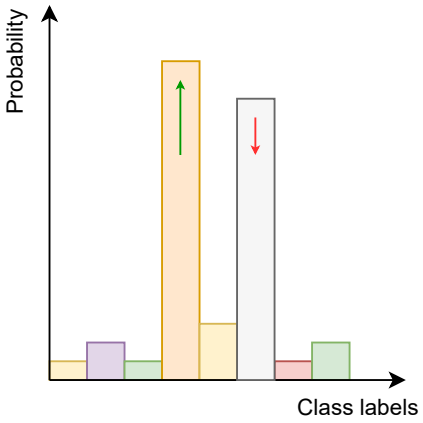


Figure 5.3: Minimum Class Confusion (MCC). The goal of MCC is to reduce the chance that the network predicts two different classes for the same sample with an equally high probability.

5.3.4 Complementary Entropy (CENT)

The *cross-entropy loss* ℓ_{CE}, typically used in supervised classification tasks, maximizes the *log-likelihood* of the predicted labels $\mathbf{y}$ with respect to the ground truth

class labels $\hat{\mathbf{y}}$, while the predictions for the erroneous classes have no role in the training process (Eq. 5.10), as they are discarded in the loss computation.

$$\ell_{CE} = -\mathbb{E}_{\hat{\mathbf{y}}}[\mathbf{y}] = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\hat{y}_{i,c}\log y_{i,c} = -\frac{1}{N}\sum_{i=1}^{N}\log y_{i,g} \tag{5.10}$$

Complementary Objective Training (COT) [86] proposes to combine cross-entropy loss with a complementary objective function that neutralizes erroneous predictions, flattening their probability distribution and thus increasing the model uncertainty about classes that do not match the ground truth. The authors of [86] propose the *Complementary Entropy (CENT)* loss function ℓ_{cent} as a secondary objective to maximize the entropy of the erroneous predictions. In mathematical terms, the CENT loss is defined as

$$\ell_{cent} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1,j\neq g}^{K}\frac{y_{i,j}}{1-y_{i,g}}\log\frac{y_{i,j}}{1-y_{i,g}} \tag{5.11}$$

where $y_{i,j}$ is the probability of sample i belonging to class j and g is the index of the ground truth label. The minimum point of the CENT loss is reached as $y_{i,j} \to (1-y_{i,g})/(K-1)$, meaning that all the erroneous labels share the same probability. As a result, CENT is expected to provide better discriminability between the ground truth and the erroneous labels.

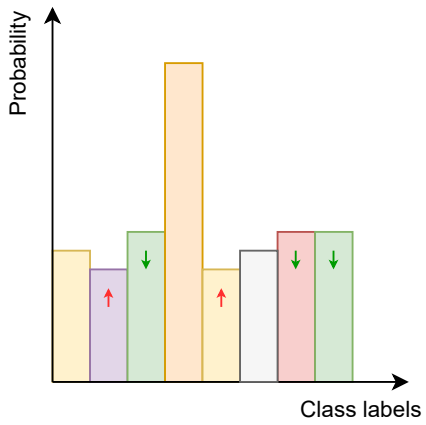


Figure 5.4: Complementary Entropy (CENT). The goal of the CENT loss is to flatten the distribution of all but the highest probability labels.

Although CENT was originally proposed for supervised scenarios where the ground truth labels are known, the approach can be easily generalized to the unsupervised context of TTA under the assumption that the ground truth is given by the network's strongest prediction.

5.3.5 TENT

Similarly to entropy minimization, TENT [87] updates the model parameters by optimizing the entropy of the predictions for the target data. However, motivated by the high dimensionality of the model parameters and by the risk of introducing unwanted noise by updating the model only on the basis of an unsupervised loss computed over the target data, TENT updates only the modulation parameters γ and β of the model normalization layers. As these parameters are applied to channels, their dimensionality is much smaller compared to the total number of model parameters. During adaptation, the model leverages the normalization statistics of the target data, as in [77, 78].

The authors of [87] also note an interesting result of TENT compared to α-BN. Indeed, while α-BN moves the activation maps closer to those seen at training time, TENT brings them closer to those produced by an oracle trained on the target data, suggesting a more task-specific behaviour. However, it can be speculated that this is mainly due to the choice of the loss function used with TENT, since unsupervised entropy minimization and supervised cross entropy minimization accomplish similar goals, especially when the domain shift is modest and the model is already able to output good predictions without adaptation to be further improved with entropy minimization.

Two minor extensions of TENT are also considered in this work. Instead of replacing normalisation statistics with new estimates from the target data, *TENT-off* reuses those collected during training. *TENT-C* updates not only the affine parameters but also the classifier.

5.4 Feature level losses

5.4.1 Relative Norm Alignment (RNA)

Extending the Relative Norm Alignment loss introduced in Sec. 4.3, the authors of [88] propose to tackle the *norm-unbalance problem* in the target domain by re-balancing the contributions of the the different modalities. Since RNA is an unsupervised loss, its formulation can be extended to the TTA setting without further modification:

$$\ell_{RNA} = \left(\frac{\mathbb{E}[h(X^{m_1})]}{\mathbb{E}[h(X^{m_2})]} - 1 \right)^2 \tag{5.12}$$

where X^{m_1} and X^{m_2} are the features extracted from two different modalities m_1 and m_2 and $h(\cdot)$ is the usual L2 norm. The RNA-C loss presented in 4.3.4 can be easily adapted to the TTA setting by using pseudo-labels to group features extracted from samples presumably belonging to the same action, as in the UDA application.

Chapter 6

Experiments

This chapter translates into experiments the theoretical analysis of Test-Time Adaptation (TTA) introduced in the previous chapter. Section 6.1 describes the experimental setting used for the evaluation of the proposed approaches for TTA. Section 6.2 introduces the problem of *domain shift* by investigating the drop in performance that occurs when a model is tested on samples from an unknown distribution. Section 6.3 explores the application of class and feature losses to improve the model predictions at test time. To verify whether TTA can also be applied in the absence of domain shift, Section 6.4 attempts to use the same techniques on seen domains. A closer look to the impact of Batch Normalization layers on TTA is provided in Section 6.5. Section 6.6 presents a multi-step extension of TTA, focusing on the trade-off between accuracy improvements and adaptation steps. In Section 6.7, the performance of TTA on multi-modal models is compared with those of *SOTA UDA models*. Finally, Section 6.8 proposes an extension of TTA to online estimation of *optical flow*.

6.1 Experimental setting

All the experiments proposed in this work are performed on the EPIC-Kitchens-55 dataset [6], following the experimental protocol defined by [69] which identifies a subset of the dataset consisting of the three largest kitchens in terms of number of labeled samples. These kitchens are referred to as D1, D2 and D3. Unless otherwise specified, models are trained on the training split of two *source* domains D_i and D_j and tested on a *target* third domain D_k. If the target domain coincide with any of the source domains, it is said to be a *seen* domain, as samples from the same distribution were seen during training. On the contrary, it is an *unseen* domain.

Input During the adaptation phase, 5 equidistant clips are sampled from the video. Each clip consists of 16 frames adjacent RGB frames and 16 optical flow

frames, computed using the TV-L1 algorithm [26], with stride 2. Depending on the length of the video the clips may overlap. The visual samples are augmented using random crops, scale jitters and horizontal flips. Audio is converted to a 256x256 matrix representing the log-spectrogram of the samples, following the procedure described by [24]. During the testing phase, the same five equidistant clips are sampled from the video but no augmentation is applied to the samples, with the exception of the central crops of the visual modalities.

Implementation Each modality is processed by a separate feature extractor and classifier. RGB and Flow feature extractors use an I3D model (Sec. 3.4.3) while Audio processing uses a BN-Inception model [18, 19] pretrained on ImageNet [24]. The extracted features are fed into a classifier, consisting of a fully connected layer, which produces the raw logits. They are then combined using a *late-fusion strategy* that averages the contributions of the different modalities.

Adaptation The target data set is processed using a batch of 32 samples, each consisting of 5 clips. Clips are divided according to their index within each sample and an adaptation step is performed for each group to update the model parameters, i.e. first all clips with index 0 are processed, then all clips with index 1 and so on. Depending on the technique used, the update may consist of the calculation of new statistics for the BN layers or the back-propagation of a loss function calculated on the model predictions or on the intermediate features. Once all the clips have been processed, the model is evaluated over the same 5 clips without any training augmentation and the predictions are averaged, as in [24].

Reproducibility Experiments were run on single NVIDIA V100 32GB gpus with PyTorch 1.11. NumPy and PyTorch random generators were initialised with seed 42 before each run and all the measurements were collected and averaged over three runs. Each experiment took slightly more than one hour on average. Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it).

6.2 Do we need Test Time Adaptation?

Table 6.1 reports the top-1 accuracy, i.e. the prediction accuracy for the ground truth class, for *seen* and *unseen* domains, highlighting the dramatic drop in performances when evaluating on the latter.[1] However, not all the modalities behave the same. RGB is clearly the worst because of its dependence on visual appearances that can easily change in different kitchens. The impact of domain shift on RGB causes its accuracy to be even worse than that of audio alone. On the contrary, audio is the most robust modality, although the drop is still quite significant. Optical flow has the best accuracy in the both the seen and unseen configurations by several percentage points, with an accuracy drop only slightly above than that of audio. Although theoretically optical flow is expected to be quite robust to domain shift as it only detects motion, practically the algorithms for its computation may be deeply affected by artifacts and motion blur. Multi-modality configurations confirm the same trend, as the combination with RGB worsen the accuracy drop of both Flow and Spec.

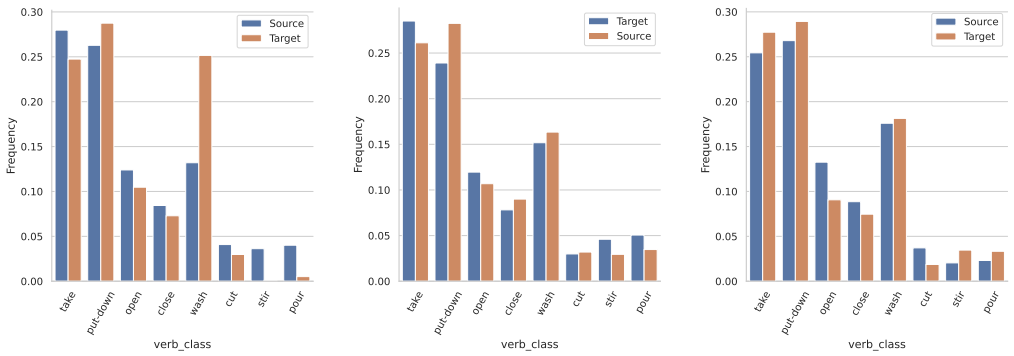
	Top-1 accuracy (%)		
	Seen domains	*Unseen* domains	Difference
RGB	53.86	36.64	▼ -17.22
Flow	**61.00**	**50.53**	▼ -10.47
Spec	52.34	43.32	▼ -9.02
RGB+Flow	**65.43**	**51.34**	▼ -14.09
RGB+Spec	61.54	51.07	▼ -10.47

Table 6.1: Top-1 accuracy of RNA-Net [1] on *seen* and *unseen* domains. As expected the accuracy drops when the source domain(s) and the target domain(s) do not coincide. Although all the modalities are severely affected by the domain gap, Flow and Spec seem to be slightly more robust compared to RGB. Flow is the best single-modality configuration and the best multi-modality configuration in combination with RGB.

6.2.1 Class imbalance

In real-world applications, the label distribution of the target dataset is not known in advance. Indeed, top-1 accuracy may not realistically capture the performance of a model in an unknown domain since it weights classes according to their size. To provide a more fair comparison of the different approaches proposed in this

[1]To limit the number of models to train, the uni-modal models presented in Table 6.1 were taken from their multi-modal counterparts and tested separately. The top-1 accuracy is expected to be close to that of uni-modal models trained from scratch, as shown by [1]. In particular, RGB and Flow models are taken from the RGB+Flow model, while the Spec model is from RGB+Spec.



(a) D1,D2→D3 **(b)** D2,D3→D1 **(c)** D3,D1→D2

Figure 6.1: Class distribution of Epic-Kitchens. The class distribution of source and target domains can vary widely and some classes may be completely absent.

work, the *per-class top-1 accuracy* measures the contribution of each class equally. Moreover, the different distribution of classes between source and destination is

	Top-1 *per-class* accuracy (%)		
	Seen domains	*Unseen* domains	Difference
RGB	48.74	26.91	▼ -21.83
Flow	**58.78**	**44.99**	▼ -13.79
Spec	33.99	31.10	▼ -2.89
RGB+Flow	**60.38**	**40.42**	▼ -19.96
RGB+Spec	54.53	39.88	▼ -14.65

Table 6.2: Top-1 *per-class* accuracy of RNA-Net [1] on seen and unseen domains. Per-class top-1 accuracy averages the accuracies of each class, regardless of their number of samples. Similarly to Table 6.1, cross-domain performances varies widely and Flow remains one of the most robust modalities, far outperforming all the other settings, even when paired with RGB.

itself part of the domain shift problem, as models tend to be more biased towards the classes they have seen more samples of.

6.3 Test Time Adaptation

This section explores the use of class and feature losses to improve model predictions, examining the accuracy gains obtained after one adaptation step over a number of domain shifts and different configurations.

6.3.1 Class losses

Class losses operate directly on the model outputs, possibly averaged over several modalities, in an attempt to improve some statistical properties of the predicted class distribution. Experiments show that different modalities exhibit quite different behaviours after one adaptation step with TTA. RGB is the worst modality in terms of domain shift, although it improves significantly to +2.91 percentage points with IM loss. However, its performance still suffers heavily from the domain shift and is inferior to that of audio alone. Flow achieves a fairly substantial improvement and is the second best model after adaptation, behind RGB+Flow by a small margin. The only modality that gets worse results after adaptation is Audio, although the drop in accuracy is quite small, -0.52 at most, suggesting that when adaptation is not effective in improving accuracy, at least it does not make it too much worse.

Multi-modal models follow a similar trend, with RGB+Flow benefiting from the individual adaptability of the two modality and RGB+Audio suffering from the poor behavior of Audio.[2] As expected, ENT loss tends to perform worse than the other techniques, as it does not take into account the diversity within each batch of data. In contrast, both MCC and IM show higher improvements, with the former far outperforming all other losses in settings where the variability of improvements is most pronounced (RGB+Audio).

Does class imbalance have a role in adaptation? The top-1 accuracy may not properly capture the class imbalance of the target dataset as it weights different classes according to their number of samples. Indeed, looking at the *per-class* accuracy from Fig. 6.4, the improvements granted by the different adaptation techniques are even more noticeable compared to the top-1 accuracy. Also, the *per-class* accuracy mitigates the poor adaptability of Audio.

TENT

The objective of the TENT losses is to minimize the entropy of the predictions even though, differently from the ENT loss, only a subset of the model parameters is

[2]The learning rates γ_{RGB} and γ_{Flow} are set to 0.001. For Audio, γ_{Audio} is set to 0.0001, with the only exception of the RGB+Audio MCC experiment in which is $\gamma_{Audio} = 0.001$.

		Top-1 accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R	N/A	36.86	34.94	38.13	36.64
	ENT (5.3.1)	39.56	37.01	41.91	39.49 (▲ +2.85)
	MCC (5.3.3)	39.56	37.01	41.96	<u>39.51</u> (▲ +2.87)
	IM (5.3.2)	39.60	37.09	41.96	**39.55** (▲ +2.91)
	CENT (5.3.4)	39.56	37.01	41.96	<u>39.51</u> (▲ +2.87)
F	N/A	47.23	51.95	52.40	50.53
	ENT (5.3.1)	49.56	51.95	57.02	52.84 (▲ +2.31)
	MCC (5.3.3)	49.62	51.95	57.07	<u>52.88</u> (▲ +2.35)
	IM (5.3.2)	49.59	51.88	57.02	52.83 (▲ +2.30)
	CENT (5.3.4)	49.56	51.95	57.16	**52.89** (▲ +2.36)
A	N/A	48.87	40.69	40.40	43.32
	ENT (5.3.1)	48.49	39.77	40.36	**42.87** (▼ -0.45)
	MCC (5.3.3)	48.53	39.92	40.04	<u>42.83</u> (▼ -0.49)
	IM (5.3.2)	48.26	39.69	40.44	42.80 (▼ -0.52)
	CENT (5.3.4)	48.49	39.77	40.13	42.80 (▼ -0.52)
R+F	N/A	46.51	48.97	58.53	51.34
	ENT (5.3.1)	51.64	50.27	58.98	<u>53.63</u> (▲ +2.29)
	MCC (5.3.3)	51.77	50.28	59.22	**53.66** (▲ +2.32)
	IM (5.3.2)	51.64	50.19	58.89	53.57 (▲ +2.23)
	CENT (5.3.4)	51.64	50.27	58.98	<u>53.63</u> (▲ +2.29)
R+A	N/A	54.93	46.67	51.60	51.07
	ENT (5.3.1)	56.13	47.20	53.42	52.25 (▲ +1.18)
	MCC (5.3.3)	56.64	47.51	53.47	**52.54** (▲ +1.47)
	IM (5.3.2)	55.82	47.20	53.60	52.21 (▲ +1.14)
	CENT (5.3.4)	56.06	47.36	53.60	<u>52.34</u> (▲ +1.27)

Table 6.3: Comparison of class-relative losses on top-1 accuracy after one adaptation step. The RGB and Flow models show the greatest improvements, even though the difference between the different adaptation techniques is very narrow. Values in **bold** and <u>underlined</u> are the best and second best values of each modality.

updated in the process. TENT and TENT-C show the most significant improvements, with accuracy close to that of the other class losses or better in the case of the Flow and RGB+Flow models. While TENT and TENT-C discard the running estimates of the BN statistics collected during training, TENT-off preserves them and results in almost insignificant improvements, hinting again that BN has a key role in adaptation. Compared to class losses (Table 6.3), RGB and Flow perform slightly better, which, given that the learning rate is small, suggests that the largest contribution comes from replacing BN running statistics with batch statistics, which makes it vulnerable to small batch sizes.

	Top-1 *per-class* accuracy (%)				
	Adaptation	D1,D2 →D3	D2,D3 →D1	D3,D1→D2	Mean
R	N/A	21.69	22.87	36.16	26.91
	ENT (5.3.1)	27.76	26.52	40.71	31.66 (▲ +4.75)
	MCC (5.3.3)	27.76	26.56	40.77	**31.69** (▲ +4.78)
	IM (5.3.2)	27.77	26.55	40.73	31.68 (▲ +4.77)
	CENT (5.3.4)	27.76	26.52	40.73	31.67 (▲ +4.76)
F	N/A	36.00	44.21	54.75	44.99
	ENT (5.3.1)	39.18	49.53	56.15	48.28 (▲ +3.29)
	MCC (5.3.3)	39.21	49.53	56.17	**48.30** (▲ +3.31)
	IM (5.3.2)	39.19	49.44	56.15	48.26 (▲ +3.27)
	CENT (5.3.4)	39.17	49.53	56.21	**48.30** (▲ +3.31)
A	N/A	28.93	23.98	40.40	31.10
	ENT (5.3.1)	29.24	23.97	40.49	31.23 (▲ +0.13)
	MCC (5.3.3)	29.27	24.04	40.49	31.26 (▲ +0.16)
	IM (5.3.2)	29.02	23.94	41.96	31.64 (▲ +0.54)
	CENT (5.3.4)	29.27	23.93	41.96	**31.72** (▲ +0.62)
R+F	N/A	27.96	36.10	57.18	40.42
	ENT (5.3.1)	34.20	43.87	56.57	44.88 (▲ +4.46)
	MCC (5.3.3)	34.47	43.87	56.59	**44.98** (▲ +4.56)
	IM (5.3.2)	34.20	43.83	56.53	44.85 (▲ +4.43)
	CENT (5.3.4)	34.20	43.87	56.57	44.88 (▲ +4.46)
R+A	N/A	34.16	35.07	50.42	39.88
	ENT (5.3.1)	40.35	37.93	45.47	**41.25** (▲ +1.37)
	MCC (5.3.3)	40.04	37.72	45.19	40.98 (▲ +1.10)
	IM (5.3.2)	38.41	37.38	45.15	40.31 (▲ +0.43)
	CENT (5.3.4)	39.87	37.50	45.31	40.89 (▲ +1.01)

Table 6.4: Comparison of class-relative losses on the top-1 *per-class* accuracy after one adaptation step. All the uni-modal models show more substantial accuracy gains compared to the standard top-1 accuracy.

		Top-1 accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R	N/A	36.86	34.94	38.13	36.64
	TENT	40.76	37.78	43.51	40.68 (▲ +4.04)
	TENT-off	36.96	34.71	38.36	36.68 (▲ +0.04)
	TENT-C	40.76	37.85	43.51	**40.71** (▲ +4.07)
F	N/A	47.23	51.95	52.40	50.53
	TENT	48.77	53.03	58.62	**53.47** (▲ +2.94)
	TENT-off	47.23	51.65	52.43	50.43 (▼ -0.10)
	TENT-C	48.70	53.03	58.62	53.45 (▲ +2.92)
A	N/A	48.87	40.69	40.40	43.32
	TENT	45.72	38.85	41.96	42.18 (▼ -1.14)
	TENT-off	48.77	40.85	40.49	**43.37** (▲ +0.05)
	TENT-C	45.72	38.70	40.49	41.64 (▼ -1.68)
R+F	N/A	46.51	48.97	58.53	51.34
	TENT	51.57	50.73	58.98	53.76 (▲ +2.42)
	TENT-off	46.61	48.97	58.53	51.37 (▲ +0.03)
	TENT-C	51.57	50.88	58.98	**53.81** (▲ +2.47)
R+A	N/A	54.93	46.67	51.60	51.07
	TENT	54.14	46.90	53.73	51.59 (▲ +0.52)
	TENT-off	54.70	46.90	51.60	51.07 (▲ +0.00)
	TENT-C	54.11	47.05	53.73	**51.63** (▲ +0.56)

Table 6.5: Comparison of TENT losses on the *top-1* accuracy after one adaptation step.

6.3.2 Feature losses

The feature level losses addressed in this section exploit the multi-modal nature of the models to improve the agreement between the features from different modalities. Since RNA showed remarkable performance in the DG and UDA settings (Sec. 5.4) by allowing the network to learn equally from multiple modalities, we can assume that the same effect can be transferred to the TTA scenario. Indeed, Tables 6.6 and 6.7 suggest that both RNA and its extension, RNA-Class, introduce improvements close to those of the class losses. Compared to RGB+Audio, the value of RNA loss on the RGB+Flow model is $\sim 10^3$ times lower, as expected since RGB and Flow frames are much closer visually than RGB and Audio representations.

		Top-1 accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R+F	N/A	46.51	48.97	58.53	51.34
	RNA	51.64	50.27	59.02	<u>53.64</u> (▲ +2.30)
	RNA-Class	51.64	50.27	59.11	**53.67** (▲ +2.33)
R+A	N/A	54.93	46.66	51.60	51.07
	RNA	56.06	47.28	53.38	**52.27** (▲ +1.20)
	RNA-Class	56.09	47.36	53.33	<u>52.26</u> (▲ +1.19)

Table 6.6: Comparison of feature losses on the top-1 accuracy. After one adaptation step, the reported feature losses achieve a top-1 accuracy close to that of the class losses (Table 6.3) for both RGB+Flow and RGB+Audio. RNA-Class provides very minor improvements over the vanilla RNA loss.

		Top-1 *per-class* accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R+F	N/A	27.96	36.10	57.19	40.42
	RNA	34.33	43.87	56.59	<u>44.93</u> (▲ +4.51)
	RNA-Class	34.33	43.87	56.63	**44.94** (▲ +4.52)
R+A	N/A	34.18	35.07	50.42	39.88
	RNA	39.85	37.51	45.22	**40.86** (▲ +0.98)
	RNA-Class	39.92	37.54	45.07	<u>40.84</u> (▲ +0.96)

Table 6.7: Comparison of feature losses on the top-1 *per-class* accuracy. Compared to the class losses shown in Table 6.3, RNA and RNA-Class perform slightly worse.

Combination of feature and class losses

As feature and class relative losses aim to solve different tasks, they may be combined together to improve the adaptation process. Table 6.7 summarizes the combination of RNA with several class relative losses.[3][4] The IM and CENT losses of the RGB+Audio model show the greatest improvement through coupling with RNA compared to the single-loss setting.

		Top-1 accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R+F	N/A	46.51	48.97	58.53	51.34
	RNA	51.64	50.27	59.02	**53.64** (▲ +2.30)
	ENT+RNA	51.64	50.27	59.02	**53.64** (▲ +2.30)
	MCC+RNA	51.64	50.27	59.02	**53.64** (▲ +2.30)
	IM+RNA	51.64	50.27	59.02	**53.64** (▲ +2.30)
	CENT+RNA	51.64	50.27	59.02	**53.64** (▲ +2.30)
R+A	N/A	54.93	46.67	51.60	51.07
	RNA	56.06	47.36	53.38	52.27 (▲ +1.20)
	ENT+RNA	56.09	47.20	53.47	52.25 (▲ +1.18)
	MCC+RNA	56.71	47.36	53.51	**52.53** (▲ +1.46)
	IM+RNA	56.06	47.28	53.64	52.24 (▲ +1.17)
	CENT+RNA	56.16	47.36	53.60	52.37 (▲ +1.30)

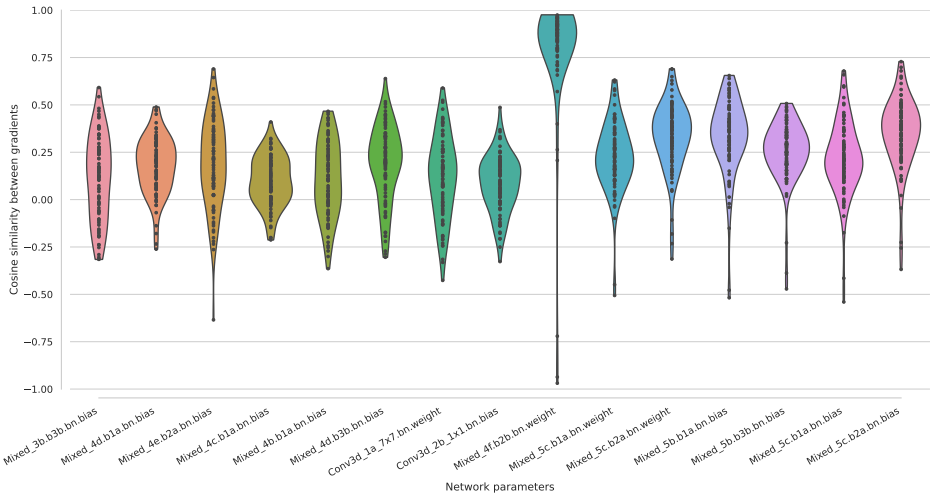
Table 6.8: Combination of class and feature losses on top-1 accuracy. Combining RNA with class relative losses results in minimal improvements for the R+F model. On the other hand, while the accuracy of R+A benefits from a larger gain.

Why does RGB+Flow show such uniform improvements? Table 6.8 shows clearly different behavior of the RGB+Flow and RGB+Audio models, with the former exhibiting completely homogeneous gains compared to the model without adaptation, suggesting that class-relative losses are not able to enhance the gains produced by RNA. To investigate the dynamics of the two losses further, Figures 6.2 and 6.3 show the cosine similarity between the gradients generated by ENT and RNA for the top 15 network parameters ranked by an importance metric that

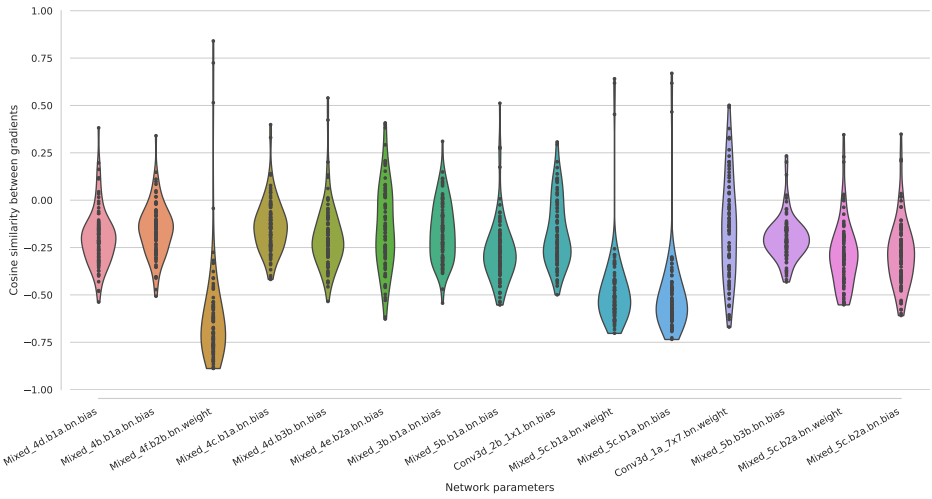
[3]For R+F models the learning rates are set to $\gamma_{RGB} = \gamma_{Flow} = 0.001$, the class relative losses have weight $w_{CRL} = 0.01$ while RNA is assigned weight $w_{RNA} = 1$.

[4]For R+A models the learning rates are $\gamma_{RGB} = 0.001$ and $\gamma_{Audio} = 0.0001$, with the only exception being the MCC+RNA configuration in which $\gamma_{Audio} = 0.001$. Both class relative losses and RNA have weights $w_{CRL} = w_{RNA} = 1$. For the IM+RNA experiment $w_{CRL} = 0.01$.

weighs more the parameters for which the gradient update is more impacting.[5][6] A



(a) RGB

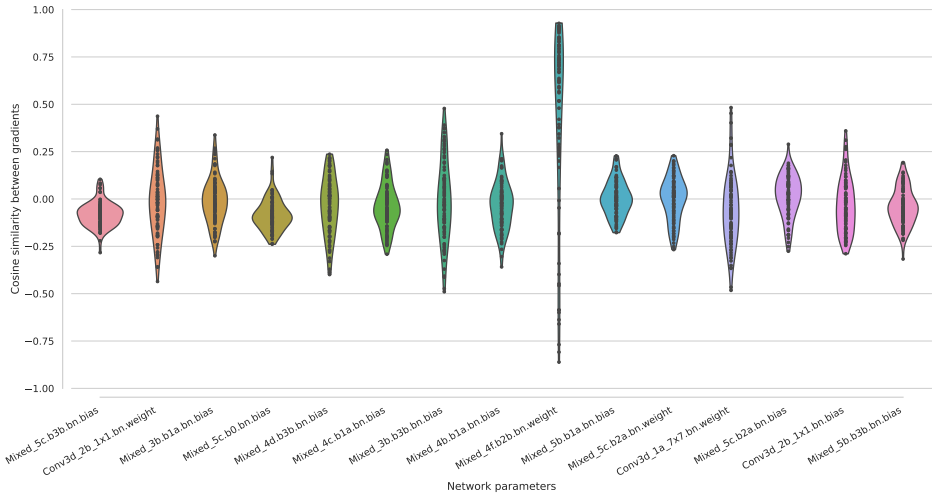


(b) Flow

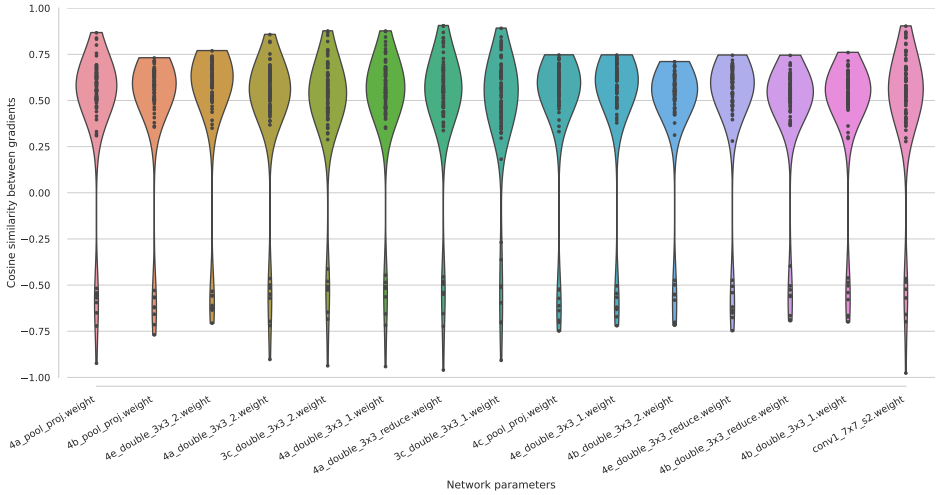
Figure 6.2: Comparison of ENT and RNA gradients (RGB+Flow). The gradients in the RGB model show mostly positive cosine similarity, as opposed to the Flow gradients, which are more skewed toward the negative region. The black dots identify the measured cosine similarity values, while the solid colored shapes model the estimated density.

[5]The importance of a parameter θ_i with respect to the loss function $\mathcal{L}$ is defined as the ratio between the L2 norm of their gradient update over the L2 norm of the parameter itself $||\frac{\partial \mathcal{L}}{\theta_i}||_2||\theta_i||_2^{-1}$.

[6]Data collected on the D1,D2→D3 shift using 1 clip per video over three runs. As in the other experiments, batch normalization layers are in training mode during the adaptation step.



(a) RGB



(b) Audio

Figure 6.3: Comparison of ENT and RNA gradients (RGB+Audio). Compared with RGB+Flow gradients, RGB here has slightly more negative gradients, although the average variability is smaller. On the other hand, Audio gradients have a strong positive similarity.

positive cosine similarity between the two gradients indicates that the two losses are cooperating to update the weights in the same direction. Conversely, a negative cosine similarity indicates that the two losses are competing with each other by updating the weights in opposite directions.

The figures suggest that RNA has a *regularization effect* on the gradients of the class losses, which is more pronounced in the RGB+Flow settings as the modality are already quite aligned, e.g. the RNA loss is small. Moreover, since in RGB+Flow

the RNA losses are small compared to the class losses, so are their gradients, resulting in minimal adjustments of the model weights that all end up in the same local minima.

In RGB+Audio, even though the RNA loss is greater than in the previous setting, being the gap between the mean feature norms of RGB and Audio more significant, the very positive cosine similarity between the two losses does not necessarily lead to an accuracy improvement for two reasons. First, the class losses are not supervised, so the direction of their gradients may or may not represent the best possible step to take.

6.4 Adapting on seen domains

Often the magnitude of the domain shift is unpredictable, and in some lucky cases the domain shift may be entirely absent when the train and test samples belong to the same domain. In fact, the definition of domain is quite loose, especially in EPIC-Kitchens, where records belong to the same domain if they were recorded in the same kitchen, even though the environments inside the same kitchen may vary widely. In addition to the common *domain shift* problem, we could introduce

		Top-1 accuracy (%)	
	Adaptation	-	*per-class*
R	N/A	53.86	48.74
	ENT (5.3.1)	<u>56.15</u> (▲ +2.29)	**53.87** (▲ +5.13)
	MCC (5.3.3)	56.14 (▲ +2.28)	**53.87** (▲ +5.13)
	IM (5.3.2)	**56.16** (▲ +2.30)	<u>53.81</u> (▲ +5.07)
	CENT (5.3.4)	56.05 (▲ +2.19)	53.13 (▲ +4.39)
F	N/A	61.00	58.77
	ENT (5.3.1)	64.43 (▲ +3.43)	**60.72** (▲ +1.94)
	MCC (5.3.3)	64.43 (▲ +3.43)	**60.72** (▲ +1.94)
	IM (5.3.2)	**64.45** (▲ +3.45)	**60.72** (▲ +1.94)
	CENT (5.3.4)	<u>64.44</u> (▲ +3.44)	<u>60.45</u> (▲ +1.87)
A	N/A	52.34	33.99
	ENT (5.3.1)	<u>53.08</u> (▲ +0.74)	34.49 (▲ +0.50)
	MCC (5.3.3)	**53.10** (▲ +0.76)	**34.57** (▲ +0.58)
	IM (5.3.2)	52.85 (▲ +0.51)	34.31 (▲ +0.32)
	CENT (5.3.4)	53.04 (▲ +0.70)	<u>34.54</u> (▲ +0.55)

Table 6.9: Comparison of the class losses on *seen* domains. Even in the absence of domain shift between source and target domain(s), TTA improves the accuracy with respect to the non adapted models. The leap is quite evident in the *per-class* accuracy, suggesting that TTA is particularly effective in adapting the models to the effective distribution of the evaluation domain.

a shallower version of domain shift that appears within the domain itself (*intra-domain shift*). As the operator moves around, the scene and background change continuously and so does the visual appearance of the frames, which is a major source of bias, especially for the RGB frames. Similarly, audio can be disturbed by completely unrelated background noises. All these aspects lead to the question of whether TTA is suitable to handle this type of domain shift or if the adaptation process risks worsening performance too much.

Experiments show that the answer is positive as TTA helps even if the domain remains the same. Table 6.9 reports the top-1 accuracy after one adaptation step with different adaptation techniques, showing that all approaches produce positive improvements, including the Audio modality.[7] Moreover, the little improvements of the latter compared to the other modalities underline the scarce adaptability of Audio which is consistent with the other TTA experiments on unseen domains (Tables 6.9 and 6.6).

6.5 The impact of Batch Normalization

Both class and feature losses show rather uniform improvements regardless of the different adaptation techniques, indicating the presence of a common ground on which all adaptation losses are based. These layers are usually put in *evaluation mode* at test time, which means that their output is calculated based on the previous estimates of the mean and variance statistics collected at training time. Depending on the similarity between the training and test samples, the use of these fixed statistics may result in activation maps that fall outside the regions seen at the time of training, possibly leading to a deterioration in performance. Figure 6.4 shows the distance between the training statistics and the online values computed at test time, across different combinations of source and target domains. When the source and target domains differ, the statistics of the current batch are further apart than the previous running estimates. In contrast, when the two domains coincide, the distance is smaller, though not zero, indicating that even within the same domains there may be discrepancies between the statistics of different samples.

How does this discrepancy affect the adaptation process? Fig. 6.5 shows the drop in accuracy after one adaptation step using the ENT loss. Regardless of modality and learning rate to adjust the size of the adaptation steps, accuracy always decreases or does not significantly improve, indicating that adaptation is detrimental in this scenario.

[7] Same configuration as Table 6.3.

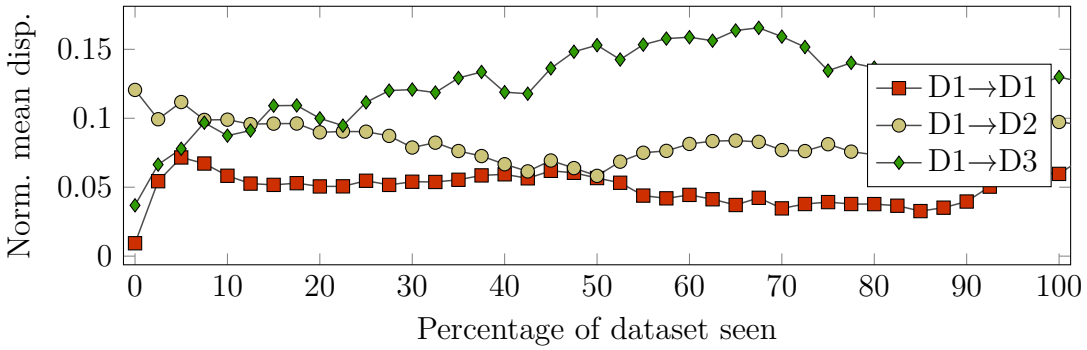


Figure 6.4: Normalized mean displacement between train and test statistics of the first BN layer in the RGB I3D model after one update of its statistics ($\lambda = 0.9$). The distance is computed as the L2-norm of the difference between the running mean estimated at training time and the updated mean after the adaptation step. The result is then normalized by dividing it by the running variance. To produce a more readable plot, distances are smoothed using a moving average of 10 samples.

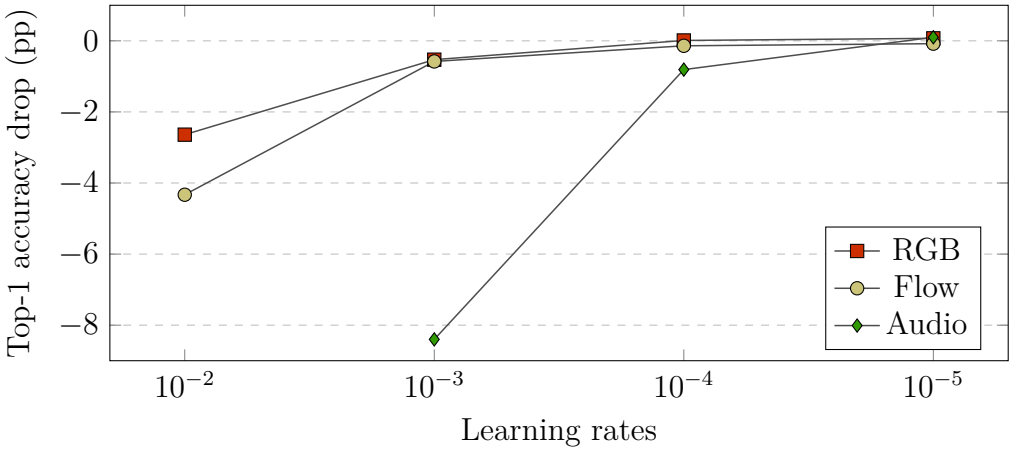


Figure 6.5: Accuracy drop without batch normalization updates after one adaptation step. The graph shows the decline in accuracy after an entropy minimization step, through a range of decreasing learning rates to adjust the magnitude of the adaptation updates. Even with small learning rates, the adaptation process results in negative or marginal improvements, suggesting that adaptation without batch normalization updates is not going to work.

6.5.1 Updating BN statistics at test time

As using the training statistics does not seem to be a viable approach for test-time adaptation, a different approach consists in updating the statistics with the new estimates computed on the test data. This can be achieved by switching the BN layer into training mode at test time, as discussed in Sec. 5.2. The momentum

hyper-parameter λ controls the magnitude of the updates.

Table 6.10 explores how this modification leads to quite significant accuracy gains in several settings, without the need for backpropagation. In addition, Fig. 6.6 compares the accuracy improvements over a fixed number of adaptation steps. When the statistics update is repeated multiple times, models develop increasing or decreasing trends depending on the value of the λ hyper-parameter. With $\lambda = 0.9$, the updated statistics tend to deviate too much from the estimates collected at training time, resulting in worse accuracy gains over time. On the contrary, with $\lambda = 0.1$ the magnitude of the updates is smaller which results in more stable improvements. Audio appears to be slightly affected by BN updates, regardless of the momentum parameter.

	Top-1 *weighted* accuracy (%)				
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
R	N/A	36.86	34.94	38.13	36.64
	BN($\lambda = 0.1$)	39.22	35.63	42.58	39.14 (▲ +2.50)
	BN($\lambda = 0.5$)	40.32	36.94	42.98	**40.08** (▲ +3.44)
	BN($\lambda = 0.9$)	39.56	37.01	41.91	39.49 (▲ +2.85)
F	N/A	47.23	51.95	52.40	50.53
	BN($\lambda = 0.1$)	48.43	54.94	56.22	**53.20** (▲ +2.67)
	BN($\lambda = 0.5$)	49.11	52.72	57.47	53.10 (▲ +2.57)
	BN($\lambda = 0.9$)	49.62	51.95	57.07	52.88 (▲ +2.35)
A	N/A	48.87	40.69	40.40	43.32
	BN($\lambda = 0.1$)	47.19	39.77	40.00	42.77 (▼ -0.55)
	BN($\lambda = 0.5$)	46.72	37.17	41.11	41.74 (▼ -1.58)
	BN($\lambda = 0.9$)	46.89	36.63	41.56	41.37 (▼ -1.95)

Table 6.10: Different performance based on the value of the momentum parameter of normalization layers. In this scenario, the batch statistics of the normalization layers are updated from the target samples and no adaptation loss is applied. λ is the momentum hyper-parameter of the BN layers. Results suggest that higher values of λ slightly degrade the accuracy of visual modalities, e.g. RGB and Flow, while any value of λ reduces the accuracy of the Audio modality.

Although BN updates are a key component of any adaptation strategy proposed in this work, they are not suitable for realistic use cases because of their dependence on the batch size to produce reliable estimates of current batch statistics. In a real-world application, possibly on a low power device, only a few samples are available at a time for TTA compared to the much large batch sizes usually used for training.

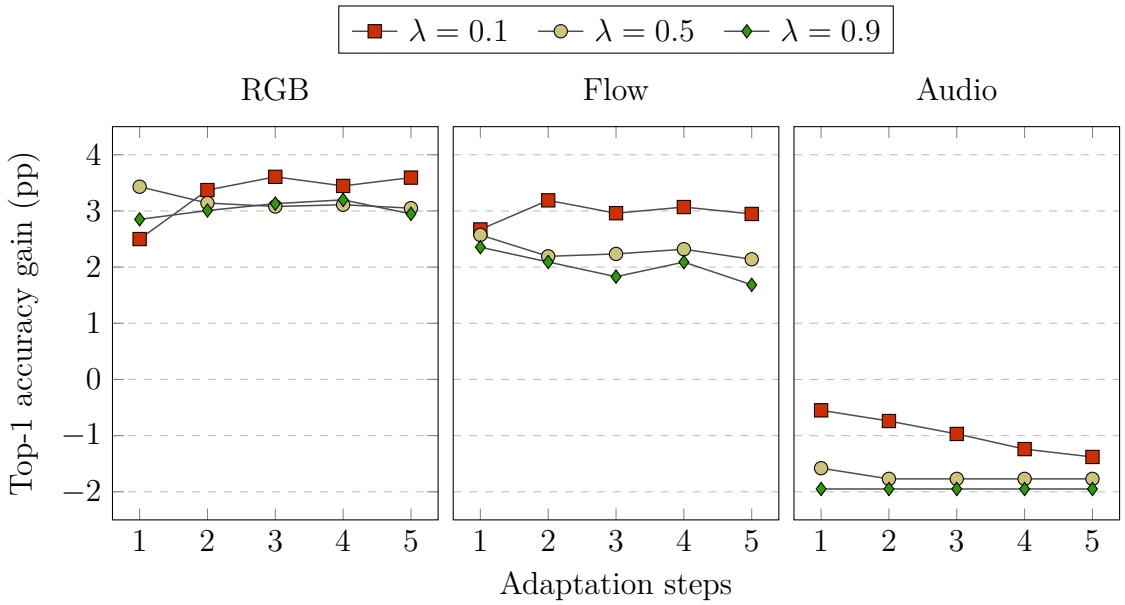


Figure 6.6: Impact on top-1 accuracy of the momentum hyper-parameter of normalization layers (multiple adaptation steps). The value of λ does not significantly impact on the accuracy of the RGB modality. Flow has the most stable trend with $\lambda = 0.1$ and decreasing performance in the other cases. The audio has inferior performance to the unadapted model for any value of λ.

6.6 Multi-step adaptation

This section extends the methods proposed in the previous sections with the possibility of repeating the adaptation step more than once before producing the final predictions. Although multi-step adaptation may not be realistically applicable due to the increased latency, it is interesting from a theoretical point of view to analyze the robustness of the adaptation techniques and the adaptability of the different modalities. Table 6.12 shows that RGB is quite robust to multi-step adaptation,

		Top-1 accuracy (%)	
	Adaptation	After 1 step	After 5 steps
R	N/A	36.64	
	ENT	39.49 (▲ +2.85)	**39.75** (▲ +3.11)
	MCC	39.51 (▲ +2.87)	39.68 (▲ +3.04)
	IM	39.55 (▲ +2.91)	39.66 (▲ +3.02)
	CENT	39.51 (▲ +2.87)	39.62 (▲ +2.98)
F	N/A	50.53	
	ENT	52.84 (▲ +2.31)	52.05 (▲ +1.52)
	MCC	52.88 (▲ +2.35)	**52.22** (▲ +1.69)
	IM	52.83 (▲ +2.30)	52.04 (▲ +1.51)
	CENT	52.89 (▲ +2.36)	52.02 (▲ +1.49)
A	N/A	43.32	
	ENT	42.87 (▼ -0.45)	**42.56** (▼ -0.76)
	MCC	42.83 (▼ -0.49)	42.21 (▼ -1.11)
	IM	42.80 (▼ -0.52)	41.95 (▼ -1.37)
	CENT	42.80 (▼ -0.52)	42.25 (▼ -1.07)

Table 6.11: Comparison of class losses on top-1 accuracy after 5 steps. RGB is the only modality to show improvements after several adaptation steps, while Flow and Audio both show decreasing trends.

with small but positive improvements after 5 steps. On the contrary, Flow and Audio exhibit a decrease in accuracy over time.

		Top-1 accuracy (%)	
	Adaptation	After 1 step	After 5 steps
R+F	N/A	51.36	
	ENT	53.63 (▲ +2.29)	53.35 (▲ +1.99)
	ENT+RNA	53.63 (▲ +2.29)	53.35 (▲ +1.99)
	MCC	**53.66** (▲ +2.32)	**53.40** (▲ +2.04)
	MCC+RNA	53.64 (▲ +2.30)	53.34 (▲ +1.98)
	IM	53.57 (▲ +2.23)	53.27 (▲ +1.93)
	IM+RNA	53.64 (▲ +2.28)	53.37 (▲ +2.03)
	CENT	53.63 (▲ +2.29)	53.34 (▲ +2.00)
	CENT+RNA	53.64 (▲ +2.30)	53.37 (▲ +2.03)
R+A	N/A	51.07	
	ENT	52.25 (▲ +1.18)	51.80 (▲ +0.73)
	ENT+RNA	52.25 (▲ +1.18)	51.71 (▲ +0.64)
	MCC	**52.54** (▲ +1.47)	**52.11** (▲ +1.04)
	MCC+RNA	52.53 (▲ +1.46)	51.58 (▲ +0.51)
	IM	52.21 (▲ +1.14)	51.41 (▲ +0.34)
	IM+RNA	52.33 (▲ +1.26)	52.08 (▲ +1.01)
	CENT	52.34 (▲ +1.27)	51.98 (▲ +0.91)
	CENT+RNA	52.37 (▲ +1.30)	51.86 (▲ +0.79)

Table 6.12: Comparison of class losses on top-1 accuracy after 5 steps. Accuracy decreases slightly after 5 steps in all cases, although the decline is more noticeable in the RGB+Audio setting when RNA is combined with the class losses.

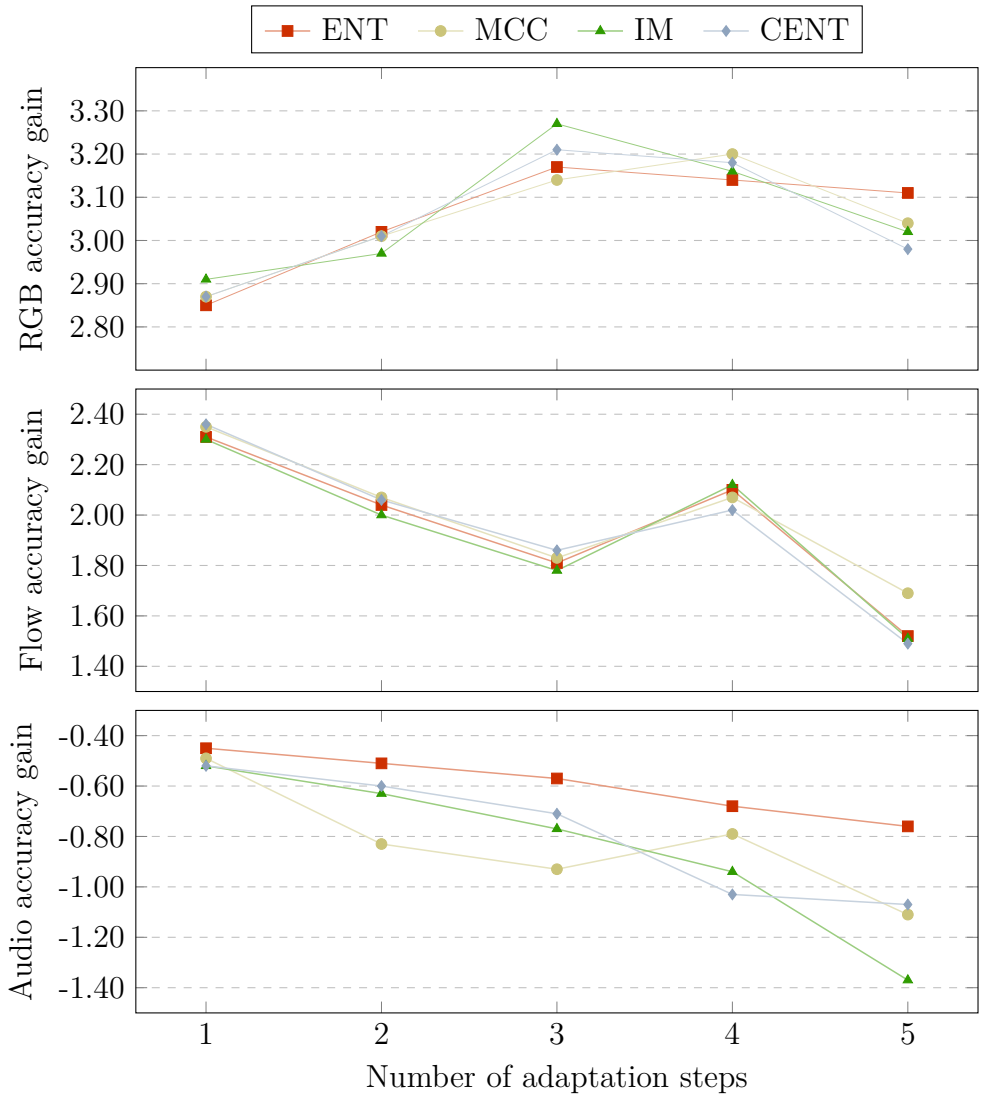


Figure 6.7: Multi-step adaptation on uni-modal models. Although there are no clear differences between the different TTA approaches, RGB has a positive trend with respect to the number of adaptation steps, while the accuracy of Flow and Audio tends to decrease with further model adaptation.

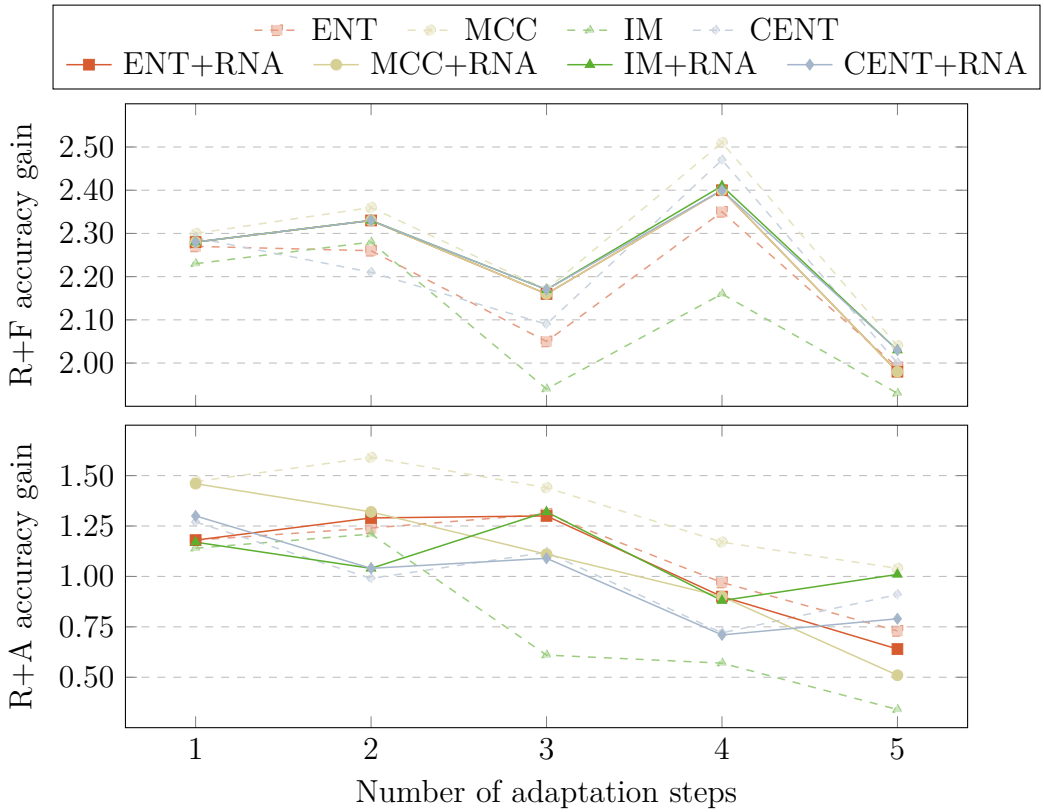


Figure 6.8: Multi-step adaptation on multi-modal models. In the RGB+Flow setting, RNA combined with class losses (solid lines) performs slightly better than class losses alone. On the other hand, all the RGB+Audio losses have a decreasing trend with no significant difference between them.

6.7 Comparison with UDA

Compared to UDA, TTA is much more limited as it exploits only a small batch of target data to adapt the model to the domain shift. On the other side, this allows TTA to adapt the model more specifically for the current batch. Therefore, the adaptation steps are based on a very narrow view of the target domain, as all updates are discarded once a new batch of data is available. Assuming we have an oracle that selects the optimal TTA strategy for each multi-modal configuration, Table 6.13 shows the best accuracy improvements obtained by adapting a multi-source RNA-Net architecture. Tables 6.14 and 6.15 list the most effective methods

Modality	**No adaptation**	**Best TTA strategy**		
RGB+Audio	51.07	MCC (1 steps)	52.54	(▲ +1.47)
RGB+Flow	51.36	MCC (4 steps)	53.85	(▲ +2.49)

Table 6.13: Best top-1 accuracy using RNA-Net (Multi-DG) + TTA.

for UDA over the subset of the EPIC-Kitchens dataset analyzed in this work and compared against models trained on one source domain only. UDA provides significant improvements over these baseline values. The DG models used as a starting

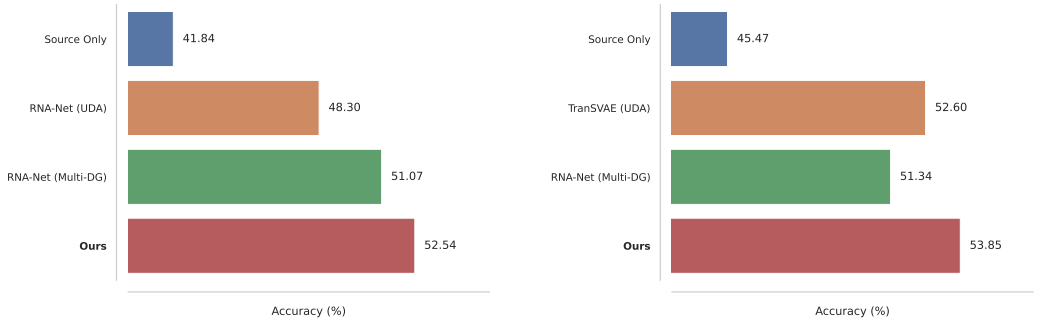
Method	**Top-1 accuracy (%)**
Source-only	41.84
MM-SADA [69]	47.75
GRL [57]	43.67
MMD [55]	44.46
AdaBN [89]	41.92
RNA-Net [1]	**48.30**

Table 6.14: Comparison of UDA methods for RGB+Audio.

Method	**Top-1 accuracy (%)**
Source-only	45.47
GRL [57]	49.40
MMD [55]	46.82
AdaBN [89]	47.20
MM-SADA [69]	50.25
Kim et al. [90]	50.98
STCDA [91]	51.20
TranSVAE [92]	**52.60**

Table 6.15: Comparison of SOTA UDA methods for RGB+Flow.

point for this work improve on the single source models, but are below UDA in the case of RGB+Audio and slightly above in the case of RGB+Flow.



(a) RGB + Audio **(b)** RGB + Flow

Figure 6.9: Accuracy improvements compared to UDA. The RGB+Audio accuracy improves by 4.24 percentage points compared to the UDA SOTA method (RNA-Net [1]. RGB+Flow improves by 1.25 percentage points over the UDA SOTA (TranSVAE [92]).

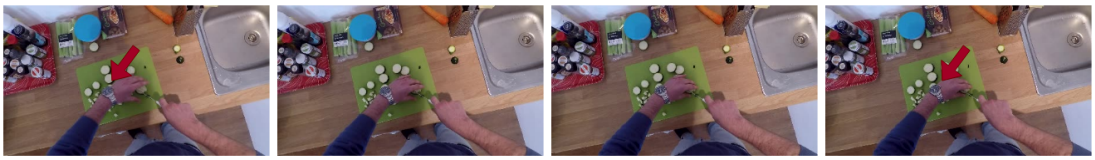
6.8 Beyond action recognition

Optical flow is a robust modality for action recognition. Experiments (Table 6.1) show that Flow alone clearly outperforms RGB and Audio modalities and comes very close to the performance of multi-modal approaches. However, the generation of optical flow frames is far from a solved problem [93]. Traditional algorithms for the computation of the optical flow from RGB frames date back to the 1981 [25]. These early methods determined the optical flow by solving complex optimisation problems based on the brightness constancy constraint, which assumes that the brightness of the scene remains constant in time and space. Besides the fact that this assumption is not always true, these methods were dramatically slow and completely unsuitable for real-time applications. In the last few years, several deep learning approaches have been proposed that learn how to solve the optical flow estimation task directly from data [93, 29, 28, 94, 4, 30]. Among these, one of the most notable CNN architectures is PWC-Net [29], which combines a light model with a processing time in the milliseconds range.

An action recognition pipeline could integrate PWC-Net to produce a rough but fast estimate of the flow at test time, while still relying on the expensive but more accurate TV-L1 [26] algorithm for training. Figure 6.10 shows the visual difference between the official TV-L1 optical flow of Epic-Kitchens [8] and the online estimate produce by PWC-Net.[8] Although the optical flow produced by the TV-L1 algorithm is by no means sharp, the arm and the hands are clearly distinguishable, which may

[8]To produce optical flow estimates that are comparable with official TV-L1 frames from Epic-Kitchens, the $i-th$ flow frame is computed from frames $2i$ and $2(i+3)$ to reduce the effect of motion blur between adjacent frames.

be sufficient for the model to understand the type of action taking place. On the other hand, the flow estimated by PWC-Net is much less accurate and part of the right arm fades into the background. A model trained on the former may struggle to recognise the same action from the latter. Furthermore, the two frames of the optical flow have a very different background, indicating an inconsistent movement that could be a further confusing element for a model. However, optical flow models



(a) Original frame sequence

(b) TV-L1

(c) PWC-Net

Figure 6.10: Optical flow frames generated with the TV-L1 algorithm and PWC-Net. The two frames show the relative motion between the second and third frames of the original sequence (the red arrows indicate the very subtle differences between the first and last frame). Although the frame is very noisy, TV-L1 allows the two arms and hands to be clearly identified, whereas the estimate produced by PWC-Net is far less accurate. TV-L1 and PWC-Net largely agree on the movement of the arm, as the colour is fairly consistent between the two frames. However, they agree less on the background, whose color shifts from pink and yellow to blue and green.

are typically trained on synthetic datasets [28, 95, 2], and applying the same weights to real-world scenarios can produce frames that are more noisy. Furthermore, the transition from TV-L1 frames to PWC-Net is itself a source of domain shift that may degrade the performance of the flow backbone of the model. Indeed, a sharp drop in accuracy is observed when testing on PWC-Net (Fig. 6.16).

Since TTA has proven to be a viable approach to improve model accuracy in the presence of domain shift, this section explores the applicability of TTA to reduce the performance drop when replacing TV-L1 frames with online estimates produced by PWC-Net. Table 6.17 shows the improvements introduced by TTA with respect to the non-adapted models. Although the accuracy is still far below the performance obtained with the TV-L1 optical flow, TTA can effectively bridge part of the gap

	Top-1 accuracy (%)	
	TV-L1 optical flow	PWC-Net estimated flow
Flow	50.53	25.04 (▼ -25.49)

Table 6.16: Performance drop at test-time with the optical flow estimated by PWC-Net on *unseen* domains. The top-1 accuracy drops dramatically.

between the two sources.

		Top-1 accuracy (%)			
	Adaptation	D1,D2→D3	D2,D3→D1	D3,D1→D2	Mean
F	N/A	47.22	51.95	52.40	50.53
F†	N/A	29.77	27.36	18.00	25.04
	ENT	31.79	32.34	32.22	32.12 (▲ +7.07)
	MCC	31.73	32.26	32.22	32.07 (▲ +7.03)
	IM	31.83	32.41	32.22	**32.15** (▲ +7.11)
	CENT	31.73	32.26	32.22	32.07 (▲ +7.03)

Table 6.17: Comparison of several TTA techniques on optical flow generated by PWC-Net. The symbol F† indicates that the optical flow is estimated using PWC-Net.

6.8.1 Improving the optical flow estimation

A more recent deep learning method for the estimation of optical flow is the FlowFormer architecture [4], featuring an *encoder-decoder structure* and *iterative refinements* of the estimated optical flow. First, given a pair of images $(I^{(1)}, I^{(2)})$, the encoder computes the cost volume, similarly to PWC-Net, and augments it using a variation of the self-attention mechanism proposed by [96]. The result, called *cost memory*, is a summary of the local and global matching patterns between the two frames. Then, the decoder transforms the cost memory into the optical flow V using recurrent refinements to reduce the distance between the flow estimated so far and the ground truth.

Like many other deep learning models for the estimation of optical flow, FlowFormer suffers from a major problem, namely the difficulty of producing clean results on real-world scenes. Indeed, these models are typically trained on synthetic datasets for which the ground truth can be easily computed as part of the rendering process [2, 97, 98], while the availability of real-world datasets [95, 99] is very limited due to the difficulty of generating proper ground truth frames. This results in lower performance when evaluating a model on real-world images, as the *domain shift* between synthetic and real images is quite significant, suggesting that the adaptation techniques proposed for action recognition could be used to improve

the quality of the estimated flow. More broadly, there seems to be a lack of studies on the application of Domain Adaptation techniques to optical flow estimation.

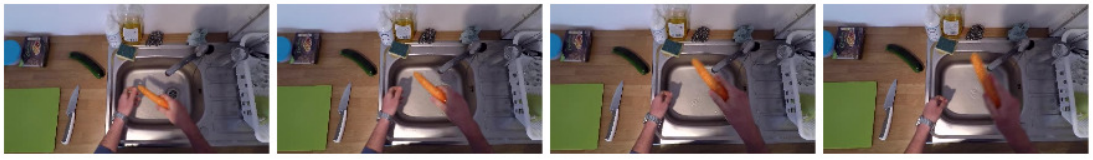
TTA for optical flow estimation

This section proposes a simple but effective application of TTA beyond the action recognition task addressed so far. Using a loss function specifically designed for optical flow estimation, TTA is able to reduce the noise introduced by FlowFormer when tested on real images from the EPIC-Kitchens dataset. Adaptation is performed online, e.g. without resetting the model after each batch, using batch size 1 and learning rate 0.0001. Samples from each video of the dataset are processed in-order. The FlowFormer model was initially pretrained on the Sintel dataset [2].

L1 smoothness loss A suitable loss function to improve the quality of the estimated optical flow is the L1 smoothness loss proposed by [30]. Given a pair of images $(I^{(1)}, I^{(2)})$ and the estimated optical flow V, the loss encourages the model to align the boundaries of the optical flow to the visual edges of the first frame.

$$\mathcal{L}_{smooth} = \frac{1}{n} \sum \left[\exp\left(-\frac{\lambda}{3} \sum_c \left| \frac{\partial I_c^{(1)}}{\partial x} \right| \right) \left| \frac{\partial V}{\partial x} \right| + \exp\left(-\frac{\lambda}{3} \sum_c \left| \frac{\partial I_c^{(1)}}{\partial y} \right| \right) \left| \frac{\partial V}{\partial y} \right| \right] \quad (6.1)$$

The first-order spatial derivatives of the first frame $I^{(1)}$ define a cost map that assigns a lower cost to pixels belonging to a visual edge, and an higher value to all the others. The cost map is then used to weigh each point of the optical flow V. If the edges of the V coincide with those of $I^{(1)}$, their associated cost is small. Otherwise, the cost is larger. By minimization of the average cost for all the points of the V, $\mathcal{L}_{smooth}$ penalizes the mismatching edges.



(a) Original frame sequence

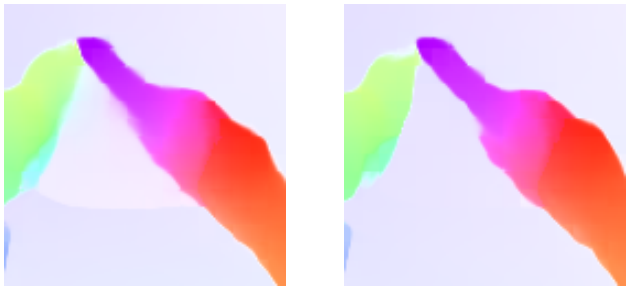
(b) TV-L1

(c) PWC-Net

(d) FlowFormer

(e) FlowFormer + TTA

Figure 6.11: Optical flow estimated using FlowFormer [4]. The optical flow frames generated with TV-L1 and PWC-Net are very noisy and low quality. FlowFormer greatly improves the estimated flow quality with sharper and cleaner edges. An adaptation step using L1 smoothness loss further improves some minor details of the flow, such as the edges of the right arm.



(a) FlowFormer **(b)** FlowFormer + TTA

Figure 6.12: Zoom of the optical flow estimated by FlowFormer in Fig. 6.12. The adaptation step sharpens the edges of the hand. The adaptation process also cancels out the central region that does not correspond to a sharp edge in the original rgb frames which may not be desirable.

Figure 6.13: Effect of TTA on the optical flow estimated by FlowFormer. The image is computed as the difference of the optical flow frames estimated by FlowFormer with and without the application of TTA (L1 smoothness loss).

Chapter 7

Conclusions

Egocentric vision is deeply affected by the domain shift problem and by the continuous changes in background, brightness and perspective produced by camera movements. In this work, we investigated an approach, Test Time Adaptation, to improve the accuracy of already robust DG models, borrowing the central idea of UDA, i.e. adaptation on target data. However, unlike UDA, TTA does not require data to be accessible during training or to train the model from scratch for each new domain, some of the main constraints imposed by UDA that make its application difficult on egocentric datasets. Instead, TTA improves the model predictions directly on test data using class and feature losses. The application of TTA techniques effectively improves the top-1 accuracy of the adapted models after just one adaptation step. However, not all domain shifts are equal, which is reflected in the different adaptability of the various modalities. The combination of RNA with other class losses indicates that RNA has a regularization effect on RGB-Flow models that allows for more stable accuracy improvements, especially when the adaptation process is repeated multiple times. TTA is able to improve the accuracy even when the domain is the same, suggesting that the dynamic and continuous adaptation provided by TTA is very effective for the action recognition task. In this respect, TTA is more versatile than UDA, as it treats each new batch of data separately, discarding updates after the prediction has been computed.

Batch Normalization (BN) layers play a key role in successful adaptation. TTA is effective provided that the BN layers update their statistics on the target samples. If this is not the case, TTA leads to zero improvements or a decrease in accuracy. Since BN updates rely on large batch sizes to estimate the statistics of the target, this constitute a large obstacle for real-world applications of TTA that needs to be investigated further in future research. Overall, starting from *multi-modal* DG models trained on multiple source domains, TTA outperforms UDA by 4.24 points on RGB+Audio and by 1.25 points when rgb is paired with optical flow. These results prove that the combination of DG techniques and TTA is an effective strategy to outperform UDA, without the strict constraints imposed by the latter.

TTA was also extended to two real-world tasks in addition to FPAR. Egocentric vision relies heavily on optical flow, which is computationally expensive. Deep learning methods for fast *optical flow estimation* [29] produce inaccurate results that may dump the performance of the Flow models. Even though most recent solutions for optical flow estimation produce high quality results [4, 94], the difference with respect to the optical flow computed by offline methods may still be substantial. TTA proved to be a viable solution to at least partially recover the performance of the model when tested on the optical flow estimates. Finally, we explored the application of TTA directly to optical flow estimation to improve the quality of the predicted frames. As *sim2real* methods for optical flow estimation are largely unexplored, this step wants to stimulate further research in this direction.

Bibliography

[1] M. Planamente, C. Plizzari, E. Alberti, and B. Caputo, "Domain generalization through audio-visual relative norm alignment in first person action recognition," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1807–1818, 2022.

[2] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)* (A. Fitzgibbon et al. (Eds.), ed.), Part IV, LNCS 7577, pp. 611–625, Springer-Verlag, Oct. 2012.

[3] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308, 2017.

[4] Z. Huang, X. Shi, C. Zhang, Q. Wang, K. C. Cheung, H. Qin, J. Dai, and H. Li, "Flowformer: A transformer architecture for optical flow," *arXiv preprint arXiv:2203.16194*, 2022.

[5] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12695–12705, 2020.

[6] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, *et al.*, "Scaling egocentric vision: The epic-kitchens dataset," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 720–736, 2018.

[7] A. Bandini and J. Zariffa, "Analysis of the hands in egocentric vision: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[8] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "The epic-kitchens dataset: Collection, challenges and baselines," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 43, no. 11, pp. 4125–4141, 2021.

[9] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11,

pp. 2278–2324, 1998.

[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[13] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*, pp. 1058–1066, PMLR, 2013.

[14] "Albumentations." `https://albumentations.ai/docs/introduction/image_augmentation/`. Accessed: 2022-10-12.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[16] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[17] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[20] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," in *International conference on machine learning*, pp. 584–592, PMLR, 2014.

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

[22] A. Cartas, J. Luque, P. Radeva, C. Segura, and M. Dimiccoli, "How much does audio matter to recognize egocentric object interactions?," *arXiv preprint arXiv:1906.00634*, 2019.

[23] A. Cartas, J. Luque, P. Radeva, C. Segura, and M. Dimiccoli, "Seeing and hearing egocentric actions: How much can we learn?," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[24] E. Kazakos, A. Nagrani, A. Zisserman, and D. Damen, "Epic-fusion: Audio-visual temporal binding for egocentric action recognition," in *Proceedings of*

the IEEE/CVF International Conference on Computer Vision, pp. 5492–5501, 2019.

[25] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[26] J. Sánchez, E. Meinhardt-Llopis, and G. Facciolo, "Tv-l1 optical flow estimation," *Image Processing On Line*, vol. 3, pp. 137–150, 07 2013.

[27] S. Savian, M. Elahi, and T. Tillo, "Optical flow estimation with deep learning, a survey on recent advances," in *Deep biometrics*, pp. 257–287, Springer, 2020.

[28] 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*, pp. 2758–2766, 2015.

[29] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8934–8943, 2018.

[30] R. Jonschkowski, A. Stone, J. T. Barron, A. Gordon, K. Konolige, and A. Angelova, "What matters in unsupervised optical flow," in *European Conference on Computer Vision*, pp. 557–572, Springer, 2020.

[31] G. Chantas, T. Gkamas, and C. Nikou, "Variational-bayes optical flow," *Journal of Mathematical Imaging and Vision*, vol. 50, 11 2014.

[32] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, *et al.*, "Event-based vision: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 1, pp. 154–180, 2020.

[33] C. Plizzari, M. Planamente, G. Goletto, M. Cannici, E. Gusso, M. Matteucci, and B. Caputo, "E2 (go) motion: Motion augmented event stream for egocentric action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19935–19947, 2022.

[34] N. Crasto, P. Weinzaepfel, K. Alahari, and C. Schmid, "Mars: Motion-augmented rgb stream for action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7882–7891, 2019.

[35] K. Min and J. J. Corso, "Integrating human gaze into attention for egocentric activity recognition," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1069–1078, 2021.

[36] Z. Zhang, D. Crandall, M. Proulx, S. Talathi, and A. Sharma, "Can gaze inform egocentric action recognition?," in *2022 Symposium on Eye Tracking Research and Applications*, pp. 1–7, 2022.

[37] G. A. Sigurdsson, A. Gupta, C. Schmid, A. Farhadi, and K. Alahari, "Charades-ego: A large-scale dataset of paired third and first person videos," *arXiv preprint arXiv:1804.09626*, 2018.

[38] Y. Li, M. Liu, and J. M. Rehg, "In the eye of beholder: Joint learning of gaze and actions in first person video," in *Proceedings of the European conference*

on computer vision (ECCV), pp. 619–635, 2018.

[39] 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*, pp. 1725–1732, 2014.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[41] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4694–4702, 2015.

[42] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional learning of spatio-temporal features," in *European conference on computer vision*, pp. 140–153, Springer, 2010.

[43] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

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

[45] G. Varol, I. Laptev, and C. Schmid, "Long-term temporal convolutions for action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 6, pp. 1510–1517, 2017.

[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[47] T. Tommasi, N. Patricia, B. Caputo, and T. Tuytelaars, "A deeper look at dataset bias," in *Domain adaptation in computer vision applications*, pp. 37–55, Springer, 2017.

[48] S. Fabbrizzi, S. Papadopoulos, E. Ntoutsi, and I. Kompatsiaris, "A survey on bias in visual datasets," *Computer Vision and Image Understanding*, p. 103552, 2022.

[49] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR 2011*, pp. 1521–1528, IEEE, 2011.

[50] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[51] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[52] Y. Zhang, "A survey of unsupervised domain adaptation for visual recognition," *arXiv preprint arXiv:2112.06745*, 2021.

[53] G. Csurka, *A Comprehensive Survey on Domain Adaptation for Visual Applications*, pp. 1–35. Cham: Springer International Publishing, 2017.

[54] R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares, *Inductive Transfer*, pp. 545–548. Boston, MA: Springer US, 2010.

[55] A. Gretton, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, K. Fukumizu, and B. K. Sriperumbudur, "Optimal kernel choice for large-scale two-sample tests," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[56] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *European conference on computer vision*, pp. 443–450, Springer, 2016.

[57] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[58] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7167–7176, 2017.

[59] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," *Advances in neural information processing systems*, vol. 29, 2016.

[60] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International conference on machine learning*, pp. 97–105, PMLR, 2015.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[62] T. Van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.

[63] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, p. 31, IEEE, 2004.

[64] Z. Meng, J. Li, Y. Gong, and B.-H. Juang, "Adversarial teacher-student learning for unsupervised domain adaptation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5949–5953, IEEE, 2018.

[65] J. Jiang, X. Wang, M. Long, and J. Wang, "Resource efficient domain adaptation," in *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 2220–2228, 2020.

[66] Y. Li, N. Wang, J. Shi, X. Hou, and J. Liu, "Adaptive batch normalization for practical domain adaptation," *Pattern Recognition*, vol. 80, pp. 109–117, 2018.

[67] W.-G. Chang, T. You, S. Seo, S. Kwak, and B. Han, "Domain-specific batch normalization for unsupervised domain adaptation," in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 7354–7362, 2019.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[69] J. Munro and D. Damen, "Multi-modal domain adaptation for fine-grained action recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 122–132, 2020.

[70] A. Sahoo, R. Shah, R. Panda, K. Saenko, and A. Das, "Contrast and mix: Temporal contrastive video domain adaptation with background mixing," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23386–23400, 2021.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[72] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[73] B. Zhou, A. Andonian, A. Oliva, and A. Torralba, "Temporal relational reasoning in videos," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 803–818, 2018.

[74] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. Yu, "Generalizing to unseen domains: A survey on domain generalization," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[75] Z. Yao, Y. Wang, J. Wang, P. Yu, and M. Long, "Videodg: generalizing temporal relations in videos to novel domains," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[76] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt, "Test-time training with self-supervision for generalization under distribution shifts," in *International conference on machine learning*, pp. 9229–9248, PMLR, 2020.

[77] Z. Nado, S. Padhy, D. Sculley, A. D'Amour, B. Lakshminarayanan, and J. Snoek, "Evaluating prediction-time batch normalization for robustness under covariate shift," *arXiv preprint arXiv:2006.10963*, 2020.

[78] S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel, and M. Bethge, "Improving robustness against common corruptions by covariate shift adaptation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11539–11551, 2020.

[79] F. You, J. Li, and Z. Zhao, "Test-time batch statistics calibration for covariate

shift," *arXiv preprint arXiv:2110.04065*, 2021.

[80] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," *Advances in neural information processing systems*, vol. 17, 2004.

[81] X. Wu, Q. Zhou, Z. Yang, C. Zhao, L. J. Latecki, *et al.*, "Entropy minimization vs. diversity maximization for domain adaptation," *arXiv preprint arXiv:2002.01690*, 2020.

[82] Y. Shi and F. Sha, "Information-theoretical learning of discriminative clusters for unsupervised domain adaptation," 2012.

[83] A. Krause, P. Perona, and R. Gomes, "Discriminative clustering by regularized information maximization," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, Curran Associates, Inc., 2010.

[84] W. Hu, T. Miyato, S. Tokui, E. Matsumoto, and M. Sugiyama, "Learning discrete representations via information maximizing self-augmented training," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 1558–1567, JMLR.org, 2017.

[85] Y. Jin, X. Wang, M. Long, and J. Wang, "Minimum class confusion for versatile domain adaptation," in *European Conference on Computer Vision*, pp. 464–480, Springer, 2020.

[86] H.-Y. Chen, P.-H. Wang, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Complement objective training," in *International Conference on Learning Representations*, 2019.

[87] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell, "Tent: Fully test-time adaptation by entropy minimization," in *International Conference on Learning Representations*, 2021.

[88] M. Plananamente, C. Plizzari, and B. Caputo, "Test-time adaptation for egocentric action recognition," in *International Conference on Image Analysis and Processing*, pp. 206–218, Springer, 2022.

[89] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, "Revisiting batch normalization for practical domain adaptation," *arXiv preprint arXiv:1603.04779*, 2016.

[90] D. Kim, Y.-H. Tsai, B. Zhuang, X. Yu, S. Sclaroff, K. Saenko, and M. Chandraker, "Learning cross-modal contrastive features for video domain adaptation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13618–13627, October 2021.

[91] X. Song, S. Zhao, J. Yang, H. Yue, P. Xu, R. Hu, and H. Chai, "Spatio-temporal contrastive domain adaptation for action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9787–9795, June 2021.

[92] X. Wang, T. Hu, X. Ren, J. Sun, K. Liu, and M. Zhang, "Transvae:a novel variational sequence-to-sequence framework for semi-supervised learning and diversity improvement," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021.

[93] M. Zhai, X. Xiang, N. Lv, and X. Kong, "Optical flow and scene flow estimation: A survey," *Pattern Recognition*, vol. 114, p. 107861, 2021.

[94] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European conference on computer vision*, pp. 402–419, Springer, 2020.

[95] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[96] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[97] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

[98] D. Sun, D. Vlasic, C. Herrmann, V. Jampani, M. Krainin, H. Chang, R. Zabih, W. T. Freeman, and C. Liu, "Autoflow: Learning a better training set for optical flow," in *CVPR*, 2021.

[99] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International journal of computer vision*, vol. 92, no. 1, pp. 1–31, 2011.