Adapting Computer Vision Models To Limitations On Input Dimensionality And Model Complexity

Alhabib Abbas

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Electronic and Electrical Engineering

University College London

February 23, 2020

I, Alhabib Abbas, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

– Alhabib Abbas

Abstract

When considering instances of distributed systems where visual sensors communicate with remote predictive models, data traffic is limited to the capacity of communication channels, and hardware limits the processing of collected data prior to transmission. We study novel methods of adapting visual inference to limitations on complexity and data availability at test time, wherever the aforementioned limitations exist. Our contributions detailed in this thesis consider both task-specific and task-generic approaches to reducing the data requirement for inference, and evaluate our proposed methods on a wide range of computer vision tasks, namely: (i) video action classification, (ii) single shot detection, and (iii) image super-resolution. Our approach studies data utility optimization techniques that omit input redundancies, and allows for the realization of data-efficient models which consider only the amount of data necessary for inference. This thesis makes four distinct contributions: (i) We investigate multi-class action classification via two-stream convolutional neural networks that directly ingest information extracted from compressed video bitstreams. We show that selective access to macroblock motion vector information provides a good low-dimensional approximation of the underlying optical flow in visual sequences. (ii) We devise a bitstream cropping method by which AVC/H.264 and H.265 bitstreams are reduced to the minimum amount of necessary elements for optical flow extraction, while maintaining compliance with codec standards. We additionally study the effect of codec rate-quality control on the sparsity and noise incurred on optical flow derived from resulting bitstreams, and do so for multiple coding standards. (iii) We demonstrate that there is a high degree of variability in the amount of data required for action classification, and leverage this to reduce the dimensionality of input volumes by inferring the required temporal extent for accurate classification prior to processing via learnable machines. (iv) We extend the paradigm of Mixtures-of-Experts (MoE) to include a new class that optimizes the data cost of inference for any computer vision task. We postulate that the minimum acceptable data cost of inference varies for different input space partitions, and consider mixtures where each expert is designed to meet a different set of constraints on input dimensionality. To take advantage of the flexibility of such mixtures in processing different input representations and modalities, we train biased gating functions such that experts requiring less information to make their inferences are favoured to others. We finally note that, our proposed data utility optimization solutions include a learnable component which considers specified priorities on the amount of information to be used prior to inference, and can be realized for any combination of tasks, modalities, and constraints on available data.

Impact Statement

The work detailed in this PhD contributed to EPSRC and InnovateUK projects, with references: (EPSRC) EP/P02243X/1, (EPSRC) EP/R025290/1, and (InnovateUK) 101932. Our study detailed in Chapter 3 was performed while keeping in mind the computational limitations of current mobile devices, and proposes low complexity video classifiers which can inform the design of future visual analysis models in contexts where limitations exist on allowed complexity. The work in Chapters 4 and 5 is on the detection of data redundancies prior to inference by computer vision models. Our contributions in that direction propose methods for complexity and data utility optimization, both of which can be used in commercial services to reduce costs of operation, or to optimize the performance of vision models that process data acquired from remote sensors. Importantly, our study in Chapter 5 considers the data utility problem generically, and its applicability is not constrained to a specific task. For example, an implementation of our contribution in Chapter 5 can be used in video streaming services, which are prioritise bandwidth-efficient transmission and compression techniques for providing content to remote playback devices. This is also true for systems that preform other vision tasks, and can be used to facilitate reduce power and bandwidth consumption in systems where the visual sensing and analysis parts are distributed across communication networks.

Acknowledgements

I would like to thank my supervisor Professor Ioannis Andreopoulos for his help and guidance. I also thank UCL, my family, and people who helped throughout my time finishing the work that went towards this PhD. Finally, I would like to thank colleagues with whom I collaborated for their work, namely: Nikolaos Deligiannis, Aaron Chadha, Yin Bi, Mohammad Jubran, and Eirina Bourtslatze.

Contents

List of Figures

List of Tables

Mathematical Notation

Care was taken to ensure the notation used in this thesis is consistent, and listed below are the most frequently used symbols and shorthand. For simplicity of presentation, some liberties were taken in notating particular parts of the thesis, and details of this are stated wherever relevant.

$\boldsymbol{v}$	a vector
v_i	i^{th} element of vector $\boldsymbol{v}$
v^k	expresses k^{th} iteration of vector $\boldsymbol{v}$ (e.g., if v^k is the k^{th} instantiation from a batch)
$\boldsymbol{x}$	an input vector
$\boldsymbol{y}$	a target vector, used to indicate outputs of predictors
$\mathcal{X}$	a set of row vectors, mostly used to represent batches of input examples
W	a matrix of weights, used to represent learnable linear transformations
$\mathcal{W}$	a set of weights, used to represent learnable weights of non-linear functions
$f(\boldsymbol{x};\theta)$	A function of $\boldsymbol{x}$ parameterized by θ, mostly used to indicate learnable functions
M	number of input examples in a batch $\mathcal{X}$
$\{x_1,...x_N\}$	a set of length N that includes variables or states $x_1,x_2,..x_N$
$\log(x)$	the logarithm function of x
$\mathcal{N}(\mu,\sum)$	multivariate drawn from a normal distribution with mean vector μ and covariance matrix $\sum$
$P(x)$	probability density function of x
$\mathbb{E}[x]$	expectation value of x, calculated as $\int xP(x)dx$

$x \sim X$	a random sample x drawn from the probability distribution function defined by X
$\lvert\boldsymbol{x}\rvert$	the l_1-norm of $\boldsymbol{x}$, which expresses $\sum_i^N \boldsymbol{x}_i$ when $\boldsymbol{x} \in \mathbb{R}^N$
$\lvert\lvert\boldsymbol{x}\rvert\rvert$	the l_2-norm of $\boldsymbol{x}$, which expresses $\sqrt{\sum_i^N \boldsymbol{x}_i^2}$ when $\boldsymbol{x} \in \mathbb{R}^N$
$\min_\theta$	minimum value of a function when searched over the set of parameters θ
$\arg\min_\theta$	instance of parameters θ that return the minimum value of a function when searched over the set of parameters θ

Chapter 1

Introduction

The fields of computer vision and learnable inference are increasingly affected by recent advances pertaining to artificial neural models, and relevant solutions are permeating the front and back ends of commercial services. This has led to the emergence of systems that facilitate the communication of whole neural architectures and their parameters [1, 2, 3], and the development of network acceleration circuits [4, 5, 6, 7, 8, 9] in recent iterations of mobile hardware [10]. In the foreseeable future, similar means of acceleration will be extended to other types of sensory hardware such as surveillance drones [11, 12, 13], and even to next generations of inter-planetary communicators [14, 15, 16].

To close the rift between the complexity requirements of deep inference models and the computational limitations of mobile devices and sensors, modern computer vision solutions are required to use modest amounts of data in producing their predictions. This is because, greedy data utility inevitably leads to more complexity and further exhaustion of communication resources (e.g., bandwidth), where both of which are scarce and typically require careful management in practical contexts. Difficulty in managing media stems primarily from two aspects: (i) all state-of-the-art methods for high-level semantic description of visual data require compute-intensive decoding, followed by complex pixel-domain processing [17, 18], and (ii) the high resolution and high frame-rate nature of decoded media and its format inflation (e.g. from standard to super-high definition, 3D, panoramic, etc) require highly-complex robust feature extraction, which imposes massive computation and storage requirements [17, 19]. The momentum of recent research in the field of computer vision is directed mainly towards achieving the maximum performance possible for a given task, and has produced a lot of promising results for many tasks such as: single shot detection [20, 21, 22], visual tracking [23, 24, 25], facial recognition [26, 27], motion prediction [28, 29], and learnable compression [30, 31]. However, recently proposed vision models ignore restrictions on complexity, and process redundantly large volumes of data. Such unnecessary burdens can be alleviated by optimizing the nature and amount of data processed by vision models without compromising performance, and this potential is backed by evidence from biological neural processes from recent studies [32, 33, 34]. Metabolic energy reserves available to the mammalian brain, for example, had important implications on its evolution and function. That is, energy consumption determines neural circuitry and activity patterns by favoring metabolically efficient wiring patterns [35, 36] and neural codes [37, 38], and to conserve energy, the human brain tends to use the minimum amount of information necessary for building higher abstractions [35, 36, 37, 38]. This observation is a strong indication that information is processed in specific ways conditional on the nature of sensed signals, and it goes inline with the energy conservation principle prevalent in most biological processes.

Like the human brain, visual inference systems are evolving to shift from previous visual analysis paradigms and standards that treat every pixel equally, to systems where information is conditionally processed subject to the context and target of inference [39, 40]. Research in this vein is mainly motivated by the chasm between the computational capabilities of mobile devices and complex vision solutions, which can be bridged in two salient directions: (i) by creating less complex solutions, and (ii) creating efficient communication infrastructures by which exemplar data can be transmitted from visual sensors to remote vision models for analysis. The aforementioned directions also motivate the thrust of our contribution, and inline with both directions, recent breakthroughs in artificial neural network design have spiked interest in developing systems that understand video content to provide content-aware compression [41, 42, 43, 44]. The latter can be helpfully instantiated in different ways for different problems, for example: media delivery systems prioritise overall picture quality, but for face detection systems [26, 27] where facial features are the prime information resource to conserve, more bitrate and transmission resources can be allocated to coding faces rather than backgrounds.

Beyond the complexity challenges mentioned above, and in visual analysis systems where visual sensors and data processing models are not co-located, limitations also exist on data transmission links (e.g., on the bandwidth of communication networks). Consequentially, this imposes constraints on the amount of data available to visual inference systems *at test time*, and in our work we denote the latter as a problem of *data utility* optimization. Examples of such systems include video streaming services which use vision models [45, 46, 47], Internet of Things (IoT) systems [48], and systems that collect data from remote sensors in general (e.g., sensors on remote drones and satellites [11, 12, 13], or inter-planetary communicators where bandwidth conservation is of utmost importance [14, 15, 16]). The need to optimize transmission links in distributed systems that include computer vision models, motivates several areas of research, namely: (i) the design of content-aware compression models that produce semantically-rich codes to incur the minimum amount of strain on transmission links, (ii) developing transmission protocols and decision mechanisms that take into account the particularities of different computer vision models, and (iii) studying heuristic and theoretical methods for approximating the required nature and amount of data for inference via computer vision models prior to transmission.

In this thesis, we start by considering video action classification to study task-specific solutions that reduce the complexity and bandwidth requirements of recent proposals to levels achievable by current mobile compute and communication systems. We then generalise our findings on video classification to extend our work to arbitrary computer vision tasks, and propose a task-agnostic solution for determining the minimum amount of data required for accurate inference. Crucially, the latter part of our work has substantial value in reducing both: the complexity of inference (because less data is processed), and required bandwidth for the acquisition of necessary data from visual sensors (because less data is sent). We begin our study on video analysis because it holds significant value as a business asset and as a tool for extracting and disseminating information. For instance, compressed video content is the prime asset of online media services such as Netflix, Amazon Prime Video, YouTube and Vimeo. The 2015-2020 Cisco Visual Networking Index Report estimates that, by 2020, more than 82 million minutes of video will be crossing the internet every second [49]. Video action classification (where human activity in video is classified, e.g. walking, swimming, etc) is a vision problem prevalent in many commercial applications such as video indexing [50, 51] and surveillance of human activity for security [52, 53]. Video requires high amounts of resources in terms of storage capacity, transmission, and computation, which motivates the need for video analysis solutions that are content-based and of manageable complexity. In many such problems, frame sequences have to be processed on visual sensors with embedded computation architectures (e.g., surveillance cameras and mobile devices [11, 12, 13]) which cannot accommodate the complexities required by such models. Our contribution on action classification considers codec motion vectors as priors of knowledge, which when properly interpreted, yield a sparse and noisy representation of the underlying motion flow in video. Motion vectors are an essential component in all current video coding standards, and are used to compress reoccurring textures across or within frames [54, 55, 56]. We show interpolated representations of motion vectors to be highly correlated with pixel-wise groundtruths of motion, with resolutions that are typically an order of magnitude lower than those of dense optical flow approximations. By training classifiers on sparse approximations of optical flow, we reduce the complexity of video classification, and show how codec motion vectors can provide a good prior for reducing the amount of texture to be processed, by including textures only when they exhibit magnitudes of motion that surpass prespecified thresholds.

In our second contribution, we consider the bitrate optimization aspect of vision models and their limits in that regard. Specifically, our contribution in this direction is motivated by contexts where the visual sensing and data processing parts of the system are distributed, where it is important to optimize the communication links between sensors and remote data processing models. To that end, and building on our first contribution, we investigate the trade-offs between video classification accuracy and compression bitrates under different quality control settings of current standard video codecs. Additionally, and to reduce bandwidth requirements between sensors and models without significantly affecting classification accuracy, we propose a learnable decision process by which frame redundancy can be inferred by inspecting low-level features of underlying motion. This can be expressed as a problem where input dimensionality must be decided before visual inference, such that redundancies can be omitted prior to transmission. Importantly, we show our decision processes to predict frame redundancy with very low complexity, such that they can be run on computation units embedded in visual sensors.

In the last part of our contribution, we consider the data utility optimization problem from a task-agnostic perspective. Specifically, we consider the extent to which input space partitions vary in the amount of information required per input in order to ensure good performance, and leverage this variance to train more data efficient mixtures of experts. To do so, we take inspiration from recent work [40, 39, 57] to propose a mixture of experts where expert utility is biased towards specific experts. While meeting predefined constraints on expert utility bias, we train a sparse gating function to select the most adequate expert to use from a set of experts of varied input requirements. Importantly, our method does not modify any pre-existing methods for complexity optimization or task specific data cost reduction. As such, our proposal can be applied in conjunction with input embedding methods [58, 59, 60, 61], or recent proposals [39, 57] that reduce the input requirements of individual experts. Crucially, when adopted for vision tasks wherein sensors and predictors are distributed, our work in this direction incurs two important benefits: (i) less complexity, because less data is used for inference, and (ii) less burden on communication resources, because less data is required to be sent between sensors and remote prediction models.

Summarised, our contribution starts by studying complexity and data utility optimization for specific classification tasks, and we ultimately generalize our findings to study both optimization problems task-agnostically. In essence, the main thrust of our thesis studies the question: "What is the minimum acceptable amount of data to use for accurate inference ?". The last contribution presented in this thesis realizes an answer to that question, and we show how useful inferences can be made about data redundancy to avert unnecessary burdens on hardware. In doing so, our contribution builds on recent breakthroughs in visual analysis to facilitate the design of data-efficient systems of computer vision and learnable inference.

Chapter 2

Literature Review

In this chapter, we review recent work published in the field of content based image and video analysis, and discuss literature that motivates and informs the details of our contributions in Chapters 3, 4, and 5. Throughout this section, we also discuss previous works which later serve as benchmarks to our proposed solutions. In the field of content based visual analysis, deep learning constitutes the de facto method upon which the current state-of-the-art is based, and we start by describing elementary design principles of artificial neural networks. Finally, we discuss specific classes of deep learning models from which our solutions to bandwidth and complexity constricted visual analysis are derived (e.g., mixtures of experts, and layer-wise conditional computing models).

2.1 Neural Networks

Neural Networks (NN) are function approximators made up of collections of acyclically connected neurons, and the simplest neural network can be modeled as a single artificial neuron. Akin to logistic regressors [62, 63, 64], artificial neurons map inputs to their output through differentiable linear transformations with learnable parameters [65]. Neural networks model complex functions as interconnected layers of artificial neurons, where outputs of lower layers are forwarded to higher layers as inputs. To approximate more complex functions, large neural networks aggregate small decisions of many neurons to model abstract notions on data, which can then facilitate high-level decisions (e.g., about whether an image belongs to a certain visual class or another). For instance, fully-connected networks - which are typically used as learnable linear transformations - define cases where all neuron pairs between adjacent layers are connected, but neurons within a layer share no connections amongst themselves [66, 67, 68]. In this way, fully-connected layers are fully visible to all subsequent layers, in the sense that: the output of any neuron in an l^{th} layer is a function of all neurons included in fully-connected layers below the l^{th} layer. Neural networks consisting only of fully-connected layers are referred to as Multi-Layer Perceptrons (MLP), especially when used as intermediate layers in more complex structures [69, 68].

2.1.1 Mathematical Representation

To formulate the function of neural networks, standard naming conventions use the term "N-layer neural network" to refer to networks with N layers *excluding* the input layer. In this way, a single-layer neural network is an architecture with no hidden layers, where the input is directly mapped to the output. A single neuron can be described as a single-layer neural network, and other learned transformations such as logistic regressors [62] and support vector machines [70] can also be expressed as a special case of single-layer neural networks (as long as an appropriate number of hidden states is allowed). The output of a single neuron can be expressed as a linear mapping which is then optionally passed to a non-linear transformation. Specifically, given an input vector $x \in R^N$, a weight vector $\boldsymbol{w} \in R^N$, and a learned bias b, the output of an artificial neuron $a(\boldsymbol{x}|\boldsymbol{w}, b)$ is:

$$a(\boldsymbol{x}|\boldsymbol{w}, b) = \sigma(\boldsymbol{w}\boldsymbol{x} + b) \tag{2.1}$$

where σ is some non-linear mapping (e.g., a hard threshold, or a sigmoid function). Activation functions take on a single number and perform a specified fixed mathematical operation, and define the non-linear aspect of artificial neurons. For instance, the sigmoid function is commonly used as a non-linearity with the mathematical form $\sigma(x) = 1/(1 + e^{-x})$. A sigmoid takes a real-valued number and limits it to be in $[0, 1]$. Importantly, sigmoid functions are differentiable which makes them suitable for training neural networks.

An undesirable property of the sigmoid non-linearity is that, when activations saturate at either tail (0 or 1), the gradient at these regions is almost zero. This makes learning weights harder once the sigmoid output saturates. Salient among common non-linearities are Rectified Linear Units (*ReLU*) which rectify some given neuron activity x as $f(x) = max(0,x)$ (i.e., activations are thresholded at zero). *ReLU* activations were found to greatly accelerate the convergence of stochastic gradient descent compared to the continuous sigmoid functions [71]. It is argued that this is due to their linear, non-saturating output. Compared to sigmoid neurons that involve expensive operations (e.g exponentials), the *ReLU* activation has a linear response and can be implemented by simply thresholding the output of a neuron at zero. Leaky rectified linear units [63] were subsequently introduced as an attempt to evade the *ReLU* zero convergence problem. Instead of the function returning zero when $x < 0$, leaky *ReLU*s instead have a small negative slope. That is, when $x < 0$ the activation is returned as $f(x) = \alpha x$ where α is a small constant, and returns $f(x) = x$ otherwise.

2.1.2 Fully-Connected Network Architectures

To express neural networks as functions, we define a set of notations that precisely refer to different components of network architectures. Using the fully-connected topology as an example, three sets are needed to fully express its function: the weights connecting each pair of neurons, the biases of each neuron, and the activations of each neuron. We formalize the notation of all constituent sets in the following, and define:

1. w^l_{jk} as the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

2. b^l_j as the bias of the j^{th} neuron in the l^{th} layer.

3. a^l_j as the activation produced by the j^{th} neuron in the l^{th} layer.

Following this notation, we can express the output of each neuron as:

$$a^l_j = \sigma(\sum_k w^l_{jk} a^{(l-1)}_j + b^l_j) \tag{2.2}$$

where $k \in \{1, 2, ..K\}$, and K is the number of visible neurons in the last layer. Note that the output of the network is a special case of a^l_j, when $l = N$.

2.1.3 Fitting Functions With Neural Networks

The process of fitting functions to neural network models is commonly referred to as "training" ; this describes the process of determining good approximations of all learnable parameters in a network (e.g., to find a^l_j and w^l_{jk} of (2.2)). In the common example of classification tasks, the last output layer is usually taken to represent classification probabilities, as arbitrary real-valued positive numbers which are necessarily summable to 1. To define a normalized probability distribution over all classes, neural network classifiers typically use the Softmax function [72, 73, 73]. Specifically, for the i^{th} class and for an example $\boldsymbol{x}$ corresponding to a ground-truth one-hot vector y, the Softmax function is expressed as:

$$P(y_i|\boldsymbol{x}) = \frac{e^{a^N_i}}{\sum_{m \neq i} e^{a^N_m}} \tag{2.3}$$

where a^N_i and a^N_i describe the i^{th} and m^{th} activations of the last layer of the network. Using the above, many of recently proposed classification networks use what is known as a cross-entropy loss for training. Loss terms are used to estimate the accuracy of predictions made by models with respect to their learned weights, and do so by calculating the cross entropy $H(y, \tilde{y}) = \mathbb{E}[-\log \tilde{y}]$ as a measure of distance between density functions. From (5.1), the cross-entropy loss term $\mathcal{L}$ is then expressed as:

$$\mathcal{L} = -\log(\frac{e^{a^N_i}}{\sum_{m \neq i} e^{a^N_m}}) \tag{2.4}$$

To find the set of weight parameters $W = [w^l_{jk}]$ that best approximates a classification function, weights are gradually updated such that L is minimized for all training examples. This is done by back-propagation, wherein the partial derivative $\frac{\partial L}{\partial w^l_{jk}}$ is considered for all weights w^l_{jk} in the network, and weights are updated accordingly [65, 66, 68], and loss is typically used in conjunction with regularization terms to impose constraints on updates made to the weight matrix W.

Neural networks with at least one hidden layer are universal approximators, and can approximate any continuous function [74]. This is to say, it can be shown that for any continuous function $f(\boldsymbol{x})$, and some acceptable error threshold $\varepsilon > 0$, there exists a neural network $g(\boldsymbol{x})$ with at least one hidden layer - and a reasonable choice of non-linearity - that achieves $\forall \boldsymbol{x} : |f(\boldsymbol{x}) - g(\boldsymbol{x})| < \varepsilon$. Neural networks fit well the statistical properties of datasets when enough data is used for training, are adequate to use in practice whenever functions to be expressed are smooth, and can be trained with different optimization algorithms that include variants of stochastic gradient descent [65, 66, 68][75]. Deeper networks that use more layers include more non-linearities, and are more capable of capturing higher level abstractions and correlations in data. We discuss limitations of the latter aspect in the following.

2.1.4 Fitting Capacities And Regularization

Allowing a network to to train with data that is too specific for certain instantiations of measurements or functions is referred to as "over-fitting", and can easily lead to a network being able to function well only under very specific conditions. The main factor that determines how much a network can learn (i.e., the capacity) is the number of layers and learnable parameters a network comprises [76, 77, 78]. That is to say, while increasing the capacity of a neural network helps the network learn more features from training data, that is not necessarily helpful, especially if there is a large discrepancy between the distributions of the training and testing data. As the features a network learns become more specific, it is less likely to generalize what it learned, and more likely to perform worse when trying to solve the problem for an input that is not very close to the examples it processed during training.

Over-fitting occurs when a model with high capacity learns many features beyond the assumed underlying structure, to the point where it starts fitting the noise in the data [19, 79]. In practice, regularization methods are used in conjunction with controlling the number of hidden layers and parameters of the network, which can be seen as a simplistic regularization method in itself. For example, by using the square magnitude of the parameters W as a penalty when calculating loss, this discourages a network from emphasizing very specific features during training. This penalty is weighted by a hyper-parameter α which specifies the amount of applied regularization. Specifically, networks with high numbers of parameters are known to have a high learning capacity, making them prone to over-fitting [19, 80, 81]. There are several techniques of imposing constraints on the weight updating process in order to prevent over–fitting and improve the generalization of functions learned by deep neural networks.. L_2 and L_1 regularization [81, 79, 19, 82, 83, 84] are common forms of regularization for learnable models in general (i.e., they are applicable to simple regressors and deep models alike). By penalizing the squared magnitude of all parameters directly in the weight update, L_2 regularization prevents weights from exploding in favor of specific features. That is, for every weight w in the network, we subtract the term αw^2 from the objective, where α is the regularization strength. The L_2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring spread-out weight vectors [79, 19]. This has the property of encouraging the network to use all of its inputs a little rather that some of its inputs a lot. Similarly in L_1 regularization, for each weight w the term $\alpha|w|$ is subtracted from the weight update. It is also possible to combine L_1 regularization with the L_2 regularization: $\alpha_1|w| + \alpha_2 w_2$, and this form of regularization is called elastic net regularization [85]. L_1 regularization has the intriguing property that it leads the weight vectors to become sparse during optimization (i.e. very close to exactly zero). In other words, neurons with L_1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to noisy inputs. Generally, L_2 regularization can be expected to give superior performance over L_1 [86, 87, 88, 84]. Another form of regularization is constraining the max-norm [89, 79], to enforce an absolute upper bound $C > 0$ on the magnitude of weight vectors for each neuron. In practice, this entails performing the parameter update normally and then enforcing the constraint by clamping the weight vector $\boldsymbol{w}$ of every neuron to satisfy $||\boldsymbol{w}||_2 < C$.

With regards to regularization techniques proposed specifically for neural network architectures, dropout [90, 91, 92, 93, 94] is a recent effective regularization technique that complements earlier methods [89, 81, 95]. While training, dropout is implemented by only keeping a neuron active with some probability $0 < p < 1$, or setting it to zero otherwise [89]. Dropout can be interpreted as sparsely sampling a neural network within the full network, and only updating parameters sampled networks after observing data. During testing there is no dropout applied (to relieve outputs from neuron sampling noise), and can be interpreted as evaluating an averaged prediction across the exponentially-sized ensemble of all sub-networks. Another recently developed technique by Szegedy et. al is Batch Normalization [96, 97, 98, 99, 80, 19, 100] alleviates a lot of problems that are caused by sampled batch biases by explicitly forcing the activations throughout a network to be normalized during training. The core observation is that this is possible because normalization is a simple differentiable operation. It has become common practice to use batch normalization in neural networks. Networks that use batch normalization are significantly more robust to bad initialization, since it eliminates batch bias [80]. Batch normalization can be interpreted as doing pre-processing at every layer of the network in a differentiable manner.

2.2 Convolving Hidden States Of Neural Networks

Convolutional Neural Networks (CNN) are a class of neural architectures wherein neurons form filters which convolve across input feature maps to produce their outputs [79, 71]. CNNs make an explicit assumption that spatial coherence is important in determining the desired output . CNNs can be expressed as fully-connected networks where weights are shared across many neuron patches that process different local regions of inputs. By sharing or tying weights across local regions of inputs, this reduces the amount of learnable parameters and correspondingly the solution space. Convolutional Layers are sets of spatial filters with learnable parameters or *weights*. Spatially, these filters are small and extend through the full depth of an input volume [79, 71]. During inference, the convolutional layer operates by *convolving* each filter across the width and height of the input to compute inner products between the weights of the filter and the input vector at all positions. This yields a 2-dimensional activation map that gives the response of that filter at every spatial position. Convolutional layers learn filters which activate upon detection a visual feature such as an edge or a blob that is statistically significant to solve the task assigned to the network. Eventually after passing through a sequence of stacked convolutional layers, a network would be able to recognize higher levels of abstraction (e.g. people, buildings). These activations are stacked along the depth dimension to produce the output volume, as illustrated in [75].

Unlike fully-connected networks, convolutional layers have neurons arranged in three dimensions: width, height, and depth (note that the word depth here refers to the third dimension of an activation volume, not to the depth of the network, which describes the total number of layers in a network). Neurons in convolutional layers are connected repeatedly across overlapping local regions of their inputs, which can be represented as tensors of any rank. CNNs gradually decrease the input volume for each layer until reaching a low-dimensional feature space that can be mapped to the final output. Within the context of multi-layered networks, convolutional layers typically define K kernels where each kernel learns a separate set features to produce the k^{th} activation map $a^k_{m,n}$, and from a feature map $\boldsymbol{f}$. Where we denote the k^{th} kernel by $\boldsymbol{h}^k$, and the indices of rows and columns of the resulting activation map are respectively marked with m and n, the convolved output is expressed as:

$$a^k_{m,n} = \sum_i \sum_j h^k_{i,j} f_{(m-i),(n-j)} \tag{2.5}$$

Note that the above can be expressed as a special case of fully-connected networks where weights are shared across local regions of input feature maps, thereby reducing the amount of operations to be performed [67, 101, 102]. In practice, models that employ convolutional layers in their function include variants of (2.5) that use "strides" to skip connections between $\boldsymbol{h}^k$ and $\boldsymbol{f}$, and use pooling layers to further reduce the dimensionality of subsequent feature maps [103]. As of yet, there is no deterministic solution to devising optimal sequences of layers of convolutional networks, although automated architecture search and design methods have been proposed [104, 105]. For instance, Zoph *et. al* [104] proposed using a dual-CNN architecture to optimize the architecture of one CNN by using the empirical differential of its loss function to update its architecture. This process is done iteratively until an optimum set of hyper-parameters is reached. This type of approach requires a large amount of computational power however, and is not commonly used. To further reduce the spatial resolution as the input propagates through the network, it is common to insert pooling layers subsequent to convolutional layers. The main motivation behind doing this is to reduce the amount of parameters and computation in the network. Adding pooling layers also helps with reducing the possibility of over-fitting, since having fewer parameters means the network has a lesser capacity to learn. Pooling layers operate independently on every depth level of the input and resize it spatially, using a nonlinear operations that selectively retain maximum values in local regions of activation maps for example. Pooling layers do not have any learnable parameters, and are entirely defined by hyper-parameters that specify the extent and means of pooling.

When considering the definitions of fully-connected layers and convolutional layers, the only difference between the two is that each neuron in a convolutional layer is connected only to a local region in the input, and that many of the neurons have the same learnable parameters (corresponding to each filter). Since the neurons in both layers simply compute products between weights and inputs, their functionality is identical, and it can be said that convolutional layers are a special-case of fully-connected layers [79]. That is to say, any convolutional layer can be expressed as a fully-connected layer, and this is actually the way by which some CNN training platforms express their architectures.

To summarize, CNN architectures [67, 19] gradually transform high dimensional inputs to lower spatial dimensions until a set of features is reached that is capable of inferring higher-level features. Notably, the now *classic* paradigm of a linear progression of cascaded layers has recently been challanged [106, 107, 108], and recent proposals feature more complicated connectivity structures. CNNs derive their value from the weight sharing aspect of their connectivity, which significantly reduces the number of learnable parameters. This, coupled with the availability of large volumes of data, eventually culminated in the realization of vision models that can accurately learn to recognize complex visual structures and semantics [67, 19, 79, 71].

2.3 State-Conditional Computing

While learnable vision models have reached a threshold of performance that allows for their adoption in commercial systems, this has spun off a new field to study methods of reducing the requirements of such models to manageable and commercially feasible levels [109, 110, 111, 112, 113, 113, 114, 115]. Some recent literature studies input compression and dimensionality optimization [61, 116], which ultimately translates to less complexity if compressed codes are used during inference, and lower bandwidth requirements for transmitting such codes. With respect to dimensionality reduction, recent proposals [117, 60, 60, 118, 119, 120] have studied specific vision tasks in order to reduce the data requirement of deep neural network models at test time. Such efforts have mainly focused on task-specific embeddings of inputs onto lower-dimensional spaces before training with more data-efficient models. This can be seen in the work of Zhang *et. al* [58] on video action classification, where the number frames used for inference is minimized by distilling temporal sequences to the most useful frames before transmitting them to remote servers for semantic analysis. Similarly, by using transfer learning, the authors in [59] show that actions can be classified accurately without the need for high-resolution optical flow, but instead by using a low-resolution optical flow approximation inferred from codec motion vectors. Similarly, recent work [121] derives a codec-specific approach for compact texture extraction from the compressed bitstream in order to reduce the data cost for action classification. Also focusing on dimensionality reduction, embedding methods such as those of [122, 20, 123, 124] can be used to produce codes that can stand in for sensed signals. This is to say, the same embeddings can be repurposed for other vision tasks as long as they retain or encode necessary information (e.g., image features for classification, or spatial coherence for localisations tasks). While mainly devised to improve the performance of image and video captioning, attention maps produced in the proposals of

[125, 108, 126, 127, 128] can also be used to reduce the data cost of inference. By highlighting important regions of inputs, attention maps also reveal redundancies which can be omitted to transfer only useful data to remote models for inference. Importantly, all the methods proposals in [59, 58] for dimensionality reduction and in [125, 108, 126] for attention pooling can only be applied for their respective tasks, and cannot be generalized to others.

Other proposals study complexity optimization directly, and propose modifications that are applicable to a wide range of models. Proposals such as static model pruning [129, 130] and MobileNets [57] reduce complexity by modifying models in a persistent manner for all inputs at test time. Other proposals use conditional computation as a way to increase model capacity while maintaining low computational costs [103, 131, 39]. Such works consider reinforcement learning and back-propagation [132, 133, 134] for training external agents to enable or disable different parts of the model by considering the unique properties of each input. For example, recent work [103, 39] proposed pruning unnecessary connections at runtime conditional on the acquired feature maps at each layer, and use reinforcement learning to determine which filters to keep in order to maintain the best performance possible. Specifically, they devise learnable agents that select kernels to use at each layer, and learn useful features from feature maps preceding each layer to optimize an objective function that balances performance with complexity. To formulate this, let C be the backbone of a CNN with convolutional layers as $C = \{\mathrm{c}_l\}_0^L$, with c_l denoting the convolutional layers whose kernels are $K_1, K_2, ..., K_M$, where convolutional layers produce feature maps $\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_M$. Methods that drop layer kernels at runtime [103, 39] do so by introducing the learnable agent $h(\boldsymbol{f}_i)$ that prunes redundant convolutional kernels in K_{i+1}, with prior knowledge of a feature map F_i in order to reduce the number of operations made at any forward-pass. To do so, after C is amply trained for some inference task, and with a with $c(\boldsymbol{f}, K)$ denoting a convolutional operation for input feature map $\boldsymbol{f}$ and kernel K, the parameters θ_h of $h(\boldsymbol{f}_i)$ are learned as:

$$\theta_h^* = argmin_{K_{i+1},h}\mathbb{E}_{\boldsymbol{f}_i}[\mathcal{L}_{cls}(c(\boldsymbol{f}_i, K[h(\boldsymbol{f}_i;\theta_h)])) + \mathcal{L}_{pnt}(h(\boldsymbol{f}_i;\theta_h))] \tag{2.6}$$

where $\mathcal{L}_{cls}$ is the loss of the classification task, $\mathcal{L}_{pnt}$ is the penalty term representing the tradeoff between the speed and the accuracy, $h(\boldsymbol{f}_i;\theta_h)$ is the conditional pruning unit that produces a list of indices of selected kernels according to input feature map, and $K[h(\boldsymbol{f}_i;\theta_h)]$ is the indexing operation that prunes kernels not selected by $h(\boldsymbol{f}_i;\theta_h)$. In optimizing (2.6), $h(\boldsymbol{f}_i)$, pruning agents such as those of [129, 130, 103, 39] are learned to account for both performance and complexity, while allowing fine-tuning of the priority of reducing complexity by including a weighing parameter w_{pnt} in $\mathcal{L}_{pnt}$. Finally and most recently, a proposal by Shazeer *et. al* [40] showed that the test-time complexity of very large networks can be significantly reduced by using sparse gating functions in mixtures of experts, where experts hold a much smaller number of weights. They take into account the fact that modern GPUs are much faster at arithmetic than at conditional branching, and show that sparse gating at the expert level can provide gains in complexity during test time. It is also important to note that, all the aforementioned works optimize solely for complexity, and always consider that the maximum amount of input to be available at test time (which will be of relevance in discussing our contributions in Chapters 3, 4, and 5).

2.4 Mixture Models And Expert Mixtures

In this section we shed light on a well studied class of probabilistic models known as Mixtures of Experts [135, 136, 137, 138, 139, 140]. Mixtures of experts are motivated by the same presuppositions that motivate other mixture models such as Gaussian Mixture Models [141] and Full Bayesian Models [142]. Specifically, expert mixtures agree with other mixture models in the way by which data instances of source distributions are viewed, namely: as the outcomes of the superposition of many simpler distributions. Mixtures of experts extend this view to mapping functions, and model desired outcomes as mixtures of different mapping functions with varying parameters.

When considering computer vision and pattern recognition applications, source distributions of inputs typically constitute a number of combined source distributions with variable properties which are well represented by mixture models [135, 136]. Expert mixtures are motivated by the fact that, when some inputs are coherent within the subspace from which they are sampled, they can be analysed using expert models designated for their specific properties. By doing so, mixtures of experts leverage the existence of properties local to different input subspaces to train corresponding experts that perform better than "know-it-all" models which view and process all inputs in the same way. By mentioning performance in the above, we refer to all different aspects of model performance, this includes: accuracy of inference, latency, and even the amount of data required by models to provide accurate results (which is ideally low, to conserve on communication network resources), and in Section 2.4.2 we discuss different ways of training mixtures to realize different types of constituent experts (e.g., specialised experts, or experts conditioned for good consensus).

To formulate the mode of operation of expert mixtures, we consider mixtures of N experts drawn from a predetermined set $\mathcal{E} = \{E_1, E_2, ..., E_N\}$, where each expert E_n is a pretrained model that attempts to determine the output $y(\boldsymbol{x})$. As long as experts map inputs to the domain of $y(\boldsymbol{x})$, they can take any predictor form (e.g., they can be simple regressors, deterministic functions, or deep neural architectures). Per input $\boldsymbol{x}$, a gating function determines the contribution of each n^{th} expert:

$$G(\boldsymbol{x}|W_g)_i = \frac{e^{f(\boldsymbol{x};W_g)_n}}{\sum_{m \neq i} e^{f(\boldsymbol{x};W_g)_m}} \tag{2.7}$$

where W_g is a set of trainable parameters of a linear transformation that defines the importance of each expert in determining $y(\boldsymbol{x})$, and $f(\boldsymbol{x};W_g) \in \mathbb{R}^N$ is the output of a specified gating model. Note that because the outermost function of $G(\boldsymbol{x};W_g)$ is a softmax transform, this necessitates that $\sum_i^N G_i(\boldsymbol{x};W_g) = 1$. The output $y(\boldsymbol{x})$ of the mixture of experts is:

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} G(\boldsymbol{x};W_g)_i E_i(\boldsymbol{x}) \tag{2.8}$$

By determining a good solution for $G(\boldsymbol{x};W_g)_i$, mixtures of mapping functions can be discovered to model probabilistic processes. In this way, expert mixtures are modeled after other mixtures [141, 142] to consider conditional processes which can be expressed as super-positions of learnable functions, where gating functions determine the importance each mapping function.

In optimizing expert mixtures, experts are commonly pre-trained prior to being included in mixtures, this is to ensure all experts are initialized reasonably and to avoid bad local minima. That is, in pretraining experts, the space of possible solutions is narrowed down to neighborhoods known to work for each expert individually (to some imminent margin of error). Subsequently, loss back-propagated through mixtures is specified as a function of the mixture output $\hat{y}(\boldsymbol{x})$ as expressed by (2.8) and some groundtruth y. We also note that, while different experts are required to process the same input, inputs can be preprocessed to accommodate any particularities that an expert may require. Additionally, a natural arbitration of mixtures of experts are hierarchical mixtures of experts, where mixtures are cascaded to provide multiple layers of conditionality that can model more complex conditional processes [143, 144, 145]. Hierarchical architectures recursively combine the outputs of cascaded mixtures to explore more complex relationships between input subspaces [144]. In Section 2.4.2 we discuss optimization methods of mixtures of experts, and in Sections 2.4.3 and 2.4.4 we discuss two subclasses of mixtures of experts to show how they are used to model and solve practical problems in computer vision.

2.4.1 Inception And Variants Of Mixtures Of Experts

The mixtures-of-experts model was first presented by Jacobs *et. al* [135] using sets of one-layer networks as experts, and was trained by using a squared error criteria to perform classification. Subsequent work on a small vowel classification problem indicated comparable performance with multi-layer perceptrons but with significantly faster training ; thereby indicating an interesting underlying property of mixtures of experts. More extensive studies on mixture of experts models were performed by Nowlan *et. al* [135], where one layer experts were substituted with multi-layer perceptrons. The work of Nowlan *et. al* [146] demonstrated that mixtures of multi-layer perceptrons were significantly more capable at generalising what is learned to unseen data. This then motivated Jacobs *et. al* [147] to show the extent of benefit provided by mixtures in two applications: combined classification, and spatial localisation of objects. In all mentioned works, mixtures were trained by non-stochastic gradient descent, and this provided a starting point for newer research to find better methods for training mixtures of experts.

2.4.2 Fitting Mixture Parameters On Data

Mixture models may be trained using a number of different techniques, and individual experts can be trained using any training method fine tuned to the source distribution from which their inputs are derived. Among others, some examples of different supervised training methods are Bayesian methods and stochastic gradient descent [147, 148, 149, 150, 151], which can also be used to train gating and experts jointly provided any ground truth to maximize the log likelihood of the mixture. Similarly, when considering unsupervised learning problems, expectation maximization [152] and adversarial learning techniques [153, 31, 154] can be used to emulate source distributions. One important aspect of training mixtures using any of the above methods is loss factorisation, which defines the way by which to quantify the contribution of each expert to the overall loss. That is to say, individual experts can be trained as: (i) specialised experts which overfit to their assigned sub-domains of data, or (ii) as parts of a larger team of experts wherein the performance of the consensus of experts is prioritised. The way in which the roles of experts is viewed is important in determining how they are trained. Depending on how mixtures are ultimately used in any application, it can be more beneficial to have "team" experts rather than "specialised" experts. Jacobs *et. al* [135, 147] give an indepth study on this specific aspect of mixtures of expert, and we summarise how each interpretation of experts reflects on the training process:

Training For Expert Consensus Where the signal used to train experts in $\mathcal{E}$ is defined such that the consensus loss is propagated back to all weights in the mixture. In effect this forces expert features to be "specialized" such that the experts compliment each other in producing the best target value possible $y(\boldsymbol{x})$.

Training For Expert Specialization Where loss is calculated not by considering the final combined output of all experts, but rather by considering the accuracy of prediction of each individual expert. Effectively this forces all experts to produce predictions closer to the ground truth, regardless of what other experts predict.

Mixtures of experts can be further classified into two sub-categories: mixtures which operate with continuous gating functions, and mixtures which selectively activate experts by using gating functions that yield sparse outputs; as a shorthand we term these two sub-categories as "soft" and "hard" gating mixtures.

2.4.3 Expert Ensembles And Boosting Methods

When considering complex domains, some problems can be better solved by considering the predictions of many pretrained models. In such instances, a straightforward solution would be to consider all possible predictions to make a final decision on some desired output, which motivated the proposition of boosting methods [155], and later on ensemble methods [155, 156]. Ensemble methods require that a set of experts are optimised before being included in a bigger mixture of experts as individual experts. Pretrained models then form an ensembles of N experts, and the final output is determined by weighted pooling of all outputs of all experts via a soft gating function. By doing so, and subject to the complexity of the gating function, more informed decisions about the output are achieved by considering each input individually to rely more on the outputs of select experts.

Ensemble mixtures typically use softmax gating to predict the final weights assigned to experts, and use loss functions that optimise the mixture gating coefficient to a least-square error. Importantly, experts are optimised individually before the gating function is trained, this is to give a distinct and valuable function for each expert. Gating functions in ensemble methods are trained for accurate consensus, since experts are optimised beforehand. Numerous studies [156, 147] have shown that ensemble methods invariably provide for better overall performance when more complex relationships exist between input sub-domains and expert suitability.

2.4.4 Selecting Experts With Sparsely Gated Mixtures

When exposed to different input samples, hard gated mixtures of experts necessarily activate only a subset of experts for inference. Eigen *et. al* [157] were the first to allude to the potential for this class of models, which would effectively turning mixtures of experts into a potential medium for conditional computing. Taking their queue from [157, 158, 103], Shazeer et al [40] are the the first work to fully explore the potential hard gated mixtures as conditional computing units. Their work does so to realize the holy grail of deep learning models that hold very large capacities while managing to conduct forward passes with feasible latency.

Similar to soft gated mixture, hard gated mixtures consist of N experts with and additional trable gating function. Importantly, the outputs of the gating function are sparsified after inference, and the authors in [40] do so for each input example by keeping the top-k gating values and setting all other gating logits to zero. Specifically, for an input $\boldsymbol{x}$ sampled from a batch of inputs $\mathcal{X}$, the gating values are calculated as the super position of a linear transformation with a weighted normal multivariate:

$$H(\boldsymbol{x})_i = \boldsymbol{x} \cdot W_g + \mathcal{N}(0,1) \cdot \psi(f(\boldsymbol{x} \cdot W_n)_i) \tag{2.9}$$

where W_{g} and W_n are learned gating parameters, $\mathcal{N}(0,1)$ is a normal multivariate with zero mean and unit variance, and $\psi(\boldsymbol{v})$ is a multivariate piecewise linear operator that returns values in v_i whenever $v_i > 0$, and returns zero otherwise. The final sparse values of the gating function by setting all $H(\boldsymbol{x})_i$ values to zero with the exception of the top-k values of $H(\boldsymbol{x})$, where k is set to be the number of retained experts for inference. Effectively, this subsequently turns off large parts of the mixture with respect to each observed input. The second term which samples weighted samples from the standard normal distribution is included to prevent load balancing issues, which we will discuss in the following section.

Sparse mixtures in the work of Shazeer *et. al.* [40] showed that with appropriate regularization and using 16 GPUs, it is possible to train hierarchical mixtures for natural language processing tasks that can hold up 68×10^9 learnable weights and process inputs with a modest 0.72 TFLOPS/GPU (and they appropriately title their work to be on "outrageously large neural networks"). To achieve this, they build on the work of [103, 157] by allowing for multiple gating decisions at many text positions, thereby compounding the number of skipped experts at each iteration. While other works modify pruning techniques for conditional computing by applying pruning at runtime, Shazeer *et. al.* [40] show that sparse gated mixtures are superior because they are trained from scratch for conditional computing. Because measuring conditionals is done exclusively by gating functions in their work, conditions are measured more sparsely than [39] where runtime pruning models measure input feature maps for every layer to determine subsequent filter activations. For these reasons, hard gated mixtures are practical templates for conditional computing in deep neural models.

2.4.5 Balancing Loads Of Sparse Mixtures

When trained only with respect to some task loss, gating functions of sparsely gated mixtures tend to converge to a state where large weight updates are always produced for the same few experts. This imbalance is self-reinforcing, and subsequently leads to some experts being favored and trained more than others (causing a lot of wasted capacity in the mixture). In other words, not constraining the training process of sparse gating functions leads to undesired local minimums that correspond to low exploitation of expert capacity, and full utilization of whatever capacity exists in a select few experts. Eigen *et. al* in [157] describe this phenomenon, and use a hard constraint to avoid local minimums, while Bengio et al. [103] include a soft constraint in the total task loss that encourages mixtures to divide batches equally between experts. To resolve this tendency, Shazeer *et. al* [40] elect to propose the best known solution to this problem by first defining the importance of an expert relative to a batch of training examples. They do so to predict the probability of selection for each expert overall for all batches, unlike [39] and [103] which consider each batch individually. Their work defines an additional loss l_{load} that estimates the batchwise sum of importance of each expert. This notion of importance takes into account how confidently experts are selected relative to other experts, and does so in a probabilistic manner such that individual batches are exemplary of other batches.

Specifically, and to balance the loads to each expert, the authors in [40] define an additional loss function to encourage experts to receive roughly equal numbers of training examples. Since the number of examples received by an expert is a discrete scalar, it cannot be used in back-propagation. To deal with this, a smooth estimator $Load(X)$ is defined as the predicted number of number of examples assigned to experts. Probability density functions are smooth and allow for back-propagating gradients back through the gating function. Denoting $H(\boldsymbol{x})_e$ as the k^{st} top value in $H(\boldsymbol{x})$ excluding the i^{th} value (where e stands in for "edge top value"), the projected future load of the i^{th} expert as predicted from an input $\boldsymbol{x}$ is denoted with $l(\boldsymbol{x})_i$, and is defined as:

$$l(\boldsymbol{x})_i = P(H(\boldsymbol{x})_i > H(\boldsymbol{x})_e) \tag{2.10}$$

and unrolling the expression yields:

$$l(\boldsymbol{x})_i = P((\boldsymbol{x} \cdot W_g)_i + \mathcal{N}(0,1)\psi(\boldsymbol{x}W_n)_i > (\boldsymbol{x} \cdot W_g)_e + \mathcal{N}(0,1)\psi(\boldsymbol{x} \cdot W_n)_e) \tag{2.11}$$

By estimating 2.11 Shazeer *et. al* [40] and others [157, 103] estimate the probability of an input $\boldsymbol{x}$ being assigned to the i^{th} expert when (2.9) is used to determine the gating output $H(\boldsymbol{x})$. The predicted load to each expert is then estimated as:

$$L(\mathcal{X})_{i=} \sum_{\boldsymbol{x}\in\mathcal{X}} l(\boldsymbol{x})_i \tag{2.12}$$

and the final load loss is:

$$\mathcal{L}_{\text{load}}(\mathcal{X})_{=} w_{\text{load}} \cdot c(L(\mathcal{X}))^2 \tag{2.13}$$

where $c(\boldsymbol{v})$ is the coefficient of variation of a vector $\boldsymbol{v}$, which is defined as the mean of $\boldsymbol{v}$ divided by its standard deviation. By defining load loss in this way, sparse gating in the mixtures of [40] are trained to divide loads equally as much as possible among constituent sets of experts, and in doing so all experts are more likely to be selected at test time to make full use of their capacity.

In summary, mixtures of experts are well studied vehicles for mixture modeling and conditional computing, and have been shown to be of great utility in many applications. Among these are boosting performance via pooling decisions made by different models [155, 156], and reducing the complexity of large capacity models by turning off large parts of the mixture subject to each observed input at test time [157, 158, 103]. Load balancing is also particularly relevant to our work in Chapter 5, where we consider the problem of deliberately maintaining imbalances of expert loads in a controlled manner (i.e., to control and bias expert utility).

2.5 Contextual Coding And Visual Compression

Within the context of engineered image compression methods, deterministic and handcrafted methods were extensively studied to fully exploit properties specific to image signals, and recently learnable image compression has attracted attention with the demonstrable success of neural architectures for image analysis [159, 42, 160]. Most salient of the recent advances in learnable image compression are auto-encoders and recurrent neural networks [161, 162, 163, 159, 42, 160]. Such networks are trained to produce latent representations to be used for image reconstruction, with the aim to minimize the mean-squared error between original and decompressed image, or to minimize a perceptual metric such as MS-SSIM [161, 162, 163, 159]. In distributed systems of visual analysis, and for the purpose of reducing demands on complexity and bandwidth, intermediary latent states of learnable machines such as autoencoders [42, 160] can be used in stead as inputs to remote inference models.

Learnable image compression [164, 165, 166] has attracted attention as the next step towards more compact image representation, and more salient among recent advances in learnable image compression are variational auto-encoders [159, 42, 160] and adversarial models[167, 31, 168]. In order to adapt learned codes to arithmetic coders, state-o-f-the-art proposals on learnable compression [165, 44, 169, 170] additionally learn context models to predict posteriors of latent code components conditional on all preceding components. Specifically, and to move learnable compression closer to replacing established coders[171, 172], context models of [170, 169] use tractable masked convolutions to regulate entropies of obtained image representations such that they can be coded more effectively by subsequent entropy coders. In distributed systems of visual analysis, and in order to reduce throughput requirements on input, intermediary latent states of learnable image reconstruction machines [159, 42, 167, 31] and entropy regulated compressors [170, 169, 164, 165] can be used instead of full-length inputs as representative signals to remote inference models.

Recent proposals also studied specific vision tasks in order to reduce the data requirement of deep neural network models at test time. For example, this can be seen in previous work [58, 59, 121] where input volumes are reduced by distilling input sequences to their most useful elements before relaying to remote servers for semantic analysis. Other work [123, 122] mainly focused on task-specific mappings of inputs onto lower-dimensional space before training with more data-efficient models, and recent advances in domain adaptation and transfer learning [173, 174, 175] can also be used to learn compressed codes tuned to particular models. However, for any specified source distribution, domain adaptation [173, 174, 175] and other proposals mentioned above [58, 59, 121] equally compact all sampled inputs to fixed length codes, and varying degrees of entropy among input examples are ignored. In this sense, while the aforementioned advances are important in determining useful transformations to enforce specific code lengths, complementary techniques are necessary to determine required code lengths prior to transformation.

The heuristic models of [165, 44, 169, 170] learn useful structures for image reconstruction directly from visual source distributions and commonly use different instances of generative models [168, 167]. Within different classes of generative models, Generative Adversarial Networks (GANs) [167, 31] have shown the greatest success in image reconstruction. Adversarial models learn a mapping from an observed image $\boldsymbol{x}$ and a random noise vector $\boldsymbol{z}$ to $G(\boldsymbol{x},\boldsymbol{z};\theta_G)$ adversarially, where discriminators distinguish generated images from images sampled directly from the source distribution that the generators try to replicate. That is, for some generative model $G(\boldsymbol{x},\boldsymbol{z};\theta_G)$ a discriminator is optimized jointly to recognise $G(\boldsymbol{x},\boldsymbol{z};\theta_G)$ among sampled examples of $\boldsymbol{x}$. This in effect trains generators towards producing images that cannot be caught out by a discriminator $D(\boldsymbol{i},\theta_D)$, such that $P(G(\boldsymbol{x},\boldsymbol{z};\theta_G))$ eventually converges to the source distribution $P(\boldsymbol{x})$. To jointly optimize $G(\boldsymbol{x},\boldsymbol{y};\theta_G)$ and $D(\boldsymbol{i};\theta_D)$. Specifically, and for some discriminator loss $\mathcal{L}(\boldsymbol{x},\boldsymbol{y};\theta_G,\theta_D)$, the generator parameters θ_G are updated to maximize $\mathcal{L}$, while the discriminator parameters θ_D are updated to minimize it. In other words, adversarial loss [167, 31] quantifies how well a discriminator $D(\boldsymbol{i};\theta_D)$ is at distinguishing generated images $G(\boldsymbol{x},\boldsymbol{y};\theta_G)$ which are not from the source $P(\boldsymbol{x})$, and is expressed as:

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{y};\theta_G,\theta_D) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}(\log D(\boldsymbol{y};\theta_D)) + \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[\log 1 - D(G(\boldsymbol{x},\boldsymbol{z};\theta_G))] \tag{2.14}$$

In training generators to match a source $P(\boldsymbol{x})$, the work of [31, 60, 120] showed that it is also beneficial to jointly optimize other losses that consider properties of $\boldsymbol{x}$ other than its saliency to $D(\boldsymbol{x};\theta_D)$, and suggest using an auxiliary loss $\mathcal{L}_1$, where:

$$\mathcal{L}_{l_1}(\boldsymbol{x},\boldsymbol{y};\theta_G) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[||\boldsymbol{y} - G(\boldsymbol{x},\boldsymbol{z};\theta_G)||_{l_1}] \tag{2.15}$$

and the learned parameters θ_G^* and θ_D^* are:

$$\theta_G^*,\theta_D^* = \arg\min_G \max_D L(\boldsymbol{x},\boldsymbol{y};\theta_G,\theta_D) + \lambda \mathcal{L}_{l_1}(\boldsymbol{x},\boldsymbol{y};\theta_G) \tag{2.16}$$

Such optimization methods have been extensively studied in recent work [167, 31] to measure the limits of structures that can be captured by such adversarially trained generators. Intermediary code mappings of $\boldsymbol{z}$ can be further finetuned to conform to prespecified properties. For example, Variational Auto Encoders (VAE) add a regularizing term such that $\boldsymbol{z}$ converges to the form of a standard normal distribution $\mathcal{N}(0,1)$. This is done to ensure that generated codes $\boldsymbol{z}$ are close in their $\mathbb{R}^n$ space, to ensure that the generators encoder does not exaggerate in expressing distances between codes z generated from closely related inputs $\boldsymbol{x}$. This is to say, whenever two inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are semantically close to each other, variational autoencoders implicity encode $\boldsymbol{z}_1$and $\boldsymbol{z}_2$ such that they are equally close while also ensuring that they are drawn from some prespecified distribution $P_V(\boldsymbol{z})$ by adding a regularization term D_{KL} to 2.16, such that it becomes:

$$\theta_G^*, \theta_D^* = \arg\min_G \max_D L(\boldsymbol{x},\boldsymbol{y};\theta_G,\theta_D) + \lambda L_1(\boldsymbol{x},\boldsymbol{y};\theta_G) - D_{KL}(P(\boldsymbol{z}|\boldsymbol{x}), P_V(\boldsymbol{z})) \quad (2.17)$$

where $P_V(\boldsymbol{z})$ is specified as the target distribution of z when $z \sim \mathcal{N}(0,1)$. Note that the condition that $\boldsymbol{z}$ has to be drawn from $\mathcal{N}(0,1)$ is enforced by a single addendum that expresses the Kullbach-Leibner distance between $\boldsymbol{z}$ and $\mathcal{N}(0,1)$.The Kullbach-Leibner distance measures the divergence between two probability distributions, and when unrolled in (2.17) the variational coding loss is:

$$\theta_G^*, \theta_D^* = \arg\min_G \max_D L(\boldsymbol{x},\boldsymbol{y};\theta_G,\theta_D) + \lambda L_1(\boldsymbol{x},\boldsymbol{y};\theta_G) - \sum_{z\in\mathbb{R}^n} P(\boldsymbol{z}|\boldsymbol{x}) \frac{P_V(\boldsymbol{z})}{P(\boldsymbol{z}|\boldsymbol{x})} \quad (2.18)$$

Latent representations of variational autoencoders are more semantically rich in that, for the same code length and when compared to their counterparts that do not conform $\boldsymbol{z}$ to take the shape to any particular distribution, codes generated by VAEs are subsequently more useful when used in image reconstruction [42, 160, 176]. Extended work on learnable image coding takes into account the entropy of latent representations in autoencoders, and to do so, the works of [161, 163] use learned context models for improved coding performance on their trained models when using adaptive arithmetic coding. To achieve the latter, for some learned code $\boldsymbol{z}$ to reconstruct the image $\tilde{\boldsymbol{x}}$ from $\boldsymbol{x}$, entropy-aware learnable coding in [161, 163] and derived works thereof, formulate loss in the form:

$$\mathcal{L} = d(\boldsymbol{x},\tilde{\boldsymbol{x}}) + \beta H(\boldsymbol{z}) \quad (2.19)$$

where $d(\boldsymbol{x},\tilde{\boldsymbol{x}})$ is some distance measure between $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$, and $H(\boldsymbol{z})$ measures the entropy of the code $\boldsymbol{z}$, and β weighs the importance of generating codes with low entropy. In [43, 177, 44], the entropy $H(\boldsymbol{z})$ is estimated by measuring the conditional probability of the i^{th} component of $\boldsymbol{z}$ conditional on all preceding components, where the probability of a code $\boldsymbol{z}$ to be sampled is:

$$p(\boldsymbol{z}) = \prod_{i=1}^{m} P(\boldsymbol{z}_i | \boldsymbol{z}_{i-1}, ...\boldsymbol{z}_1) \tag{2.20}$$

and its entropy is measured with the context model $H(\boldsymbol{z})$ as:

$$H(\boldsymbol{z}) = \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})} \sum_{i=1}^{m} -log(P(\boldsymbol{z}_i | \boldsymbol{z}_{i-1}, ...\boldsymbol{z}_1)) \tag{2.21}$$

Note that $H(\boldsymbol{z})$ can interpreted as the sum of the predictability of $\boldsymbol{z}_i$ with prior knowledge of all preceding components in $\boldsymbol{z}$. Minimizing (2.19) when training autoencoders and generative models draws the system towards low or high average entropies $H(\boldsymbol{z})$ subject to weights assigned to β. In this way, recent advances in learnable coding models [161, 162, 163, 159] use context models to produce latent representations that make it easier for subsequent entropy compression to reduce the size of representations further (by taking into account both the entropy of $\boldsymbol{z}$ in addition to its usefulness in estimating $\boldsymbol{x}$). In the context of distributed visual systems, and when generating codes to be transmitted through communication networks before inference, variational autoencoders trained adversarially can be useful as learnable compressors to produce bandwidth-efficient compact codes for image reconstruction in remote machines.

2.6 Multi-Class Action Classification

Multi-class action classification is the visual task of distinguishing sequences with respect to actions performed in human-comprising content. State of the art methods in this field [178, 179, 180, 181, 182] all agree in one aspect, namely: taking full advantage of a spatial mode that processes texture in video for inferring classes, and a spatio-temporal mode to infer actions from underlying motion structures. To do so, recent well-performing proposals [183, 184, 185, 17] use two-stream architectures and dedicate each stream to a spatial or spatio-temporal mode. The term "two-stream" here is used in reference to separated branches in model architectures, the inferences of which are ultimately merged (e.g., by averaging their predictions). In earlier work, and to first show the importance of including a temporal modality for video classification, Simonyan *et. al* in [17] argue that exclusively using volumes of RGB frames does not effectively represent motion information to CNNs. They propose using a 2D architecture with dense optical flow to represent the temporal component of the video. Notably, this temporal CNN is shown to outperform an equivalent 2D spatial stream ingesting RGB frames. They then show that performance can be improved further by fusing the spatio-temporal and spatial streams using a simple score averaging, and their two stream architecture achieved 88.0% on UCF-101 (which is a current standard benchmark for video action classification). Nevertheless, the computational cost remained high due to the requirement to extract Brox optical flow [186] for the temporal stream.

Later efforts to improve classification resulted in the work of Karpathy *et. al* [183], who proposed extending the CNN architecture from image to video by performing spatio-temporal convolutions in the first convolutional layers over a 4D video chunk $V \in \mathbb{R}^{W\times H\times K\times T}$, where W, H are the spatial dimensions, K is the number of channels and T is the number of frames in the chunk. This is the premise behind what is termed as a slow-fusion architecture, which uses 3D convolutions on RGB frame chunks in the first 3 layers, thus encompassing the full spatio-temporal extent of the input. Notably, experiments demonstrated that feeding a single RGB frame versus multiple frames into this architecture did not have any significant effect on accuracy; in other words, the CNN was not effectively learning on the motion information. Tran *et. al* [187] improve accuracy by using a deep 3D CNN VGGnet architecture [188] together with spatio-temporal convolutions. They combine this with a bagging approach over 3 networks to achieve a state-of-the-art accuracy of 90.4% on UCF-101 [184], albeit with heavy computational cost.

2.6.1 Action Classification In The Compressed Domain

To avert complexity overheads of calculating optical flow as a spatio-temporal modality, the use of codec motion vectors as approximations of optical flow was first proposed for action recognition by Kantorov and Laptev [189]. Their approach preceded the surge in convolutional neural networks for image classification and used Fisher vectors (which achieve lower accuracies in standard action recognition datasets) in their stead. More recently, Zhang *et. al* [190] utilized codec motion vectors as input to a 2D CNN for action recognition with a framework that requires optical-flow based training and transfer learning [191]. Their requirement of highly-upsampled frames during inference increases the implementation complexity, as large activation maps need to be calculated at the first layers of their CNN. Recent work [121, 192], among which is some of our contributions, showed that compressed-domain action recognition can achieve accuracy that competes with optical-flow based methods, while offering higher ingestion and CNN processing speed than all previous alternatives. Given that the spatial stream learns on scene information that tends to be persistent across frames, compressed-domain methods gain by sparse frame decoding combined with motion-adaptive super-positioning of decoded macroblock information to generate intermediate frames at a finer temporal scale. Recent work [121, 192] showed that compressed-domain action recognition can achieve accuracy that competes with optical-flow based methods, while offering higher ingestion and CNN processing speed than all previous alternatives. Given that the spatial stream learns on scene information that tends to be persistent across frames, compressed-domain methods gain by sparse frame decoding combined with motion-adaptive super-positioning of decoded macroblock information to generate intermediate frames at a finer temporal scale.

One of the issues with most of the work described above is the short temporal extent of inputs [121, 193, 194]; each input video segment comprises a small group of frames that only represent (approximately) one second of the recorded action or event to be classified. Hence, this cannot account for cases where temporal dependencies extend over longer durations [121]. Feichtenhofer *et al.* [185] attempted to resolve this issue by using multiple copies of their two stream network where the copies are spread over a coarse temporal scale, thus encompassing both coarse and fine motion information with an optical flow input. Despite achieving state-of-the-art results on UCF-101 and HMDB-51 datasets, their approach requires heavy processing for both training and testing. Other work [195, 18] argues that increasing the temporal extent is simply a case of taking the optical flow component over a larger temporal extent. In order to minimize the complexity of the network, most such approaches downsize frames in order to reduce spatial dimensions. This is now increasingly important due to the advent of visual IoT and cloud-based platforms, where the visual sensing and processing are not co-located [196, 197, 198]. Alas, such tradeoffs are non trivial, because they depend on the spatio-temporal information needed by the CNN performing the recognition task [195, 199]. On the other hand, the work of Sevilla *et al.* [29] shows that high-resolution optical flow can be beneficial since deep learning methods can learn features from small details. This observation suggests that high-resolution optical flow can be leveraged to lower the temporal extent of inputs. Following the latter in our recent work [200], and to omit redundancies in video bitstreams, we studied the trade-offs in compressed-domain spatio-temporal information and explored the rate-accuracy characteristics of CNN-based video action classification.

2.7 Contribution Outline and Research Outcomes

The remaining chapters of this thesis detail our contributions on three fronts: (i) visual classification in the compressed domain, (ii) a study of the relationship between technical aspects of video coding and their effects on subsequent classification accuracy, and (iii) a task-agnostic study on the relationship between data utility and inference accuracy, in which we realize a novel class of mixtures of experts to optimize the performance of any computer vision model under any constraint on allowed limits of data use. In this way, our work follows a neat trajectory, where we begin by considering the trade-offs between accuracy, complexity, and bitrate for video classification specifically, and we finally generalize our findings to allow for the exploitation of computer vision models under different constraints on communication bandwidth and complexity commonly found in practice.

In Chapter 3, and inspired by recent breakthroughs made in video classification, we consider the problem of ultra-fast classification that would allow for the classification of videos in realtime. While the best preforming video classification models infer classes from the pixel-domain, and typically augment their input sources with optical flow, we adopt a minimalist approach to data acquisition and utility to train computer vision models on the compressed domain directly. In our work we consider the video encoder as an imperfect-yet-highly-efficient sensor that derives spatio-temporal activity representations with minimal processing. This is to say, in our contribution the data is acquired without decompression, and is simply read from the bitstream and processed to infer a sparse approximation of optical flow. Our work focuses on complexity reduction and we show that video classifiers based on deep learning methods can indeed capture useful features from such approximations to produce classes with an accuracy comparable to that of models that use fully decompressed pixel frames for inference. We also extend our method to preform selective processing of textures in frames, and show that this can also provide a modest increase in classification accuracy. Our results show that direct inference from the compressed domain comes at a marginal loss in accuracy, and provides high gains in speed when compared to conventional deep neural networks trained on RGB frames and dense optical flow estimations.

In Chapter 4, we direct our attention to minimizing the required bitrate for accurate action classification, and we do this to explore the potential for removing more redundancies in compressed video data. By inspecting simple statistics of motion vectors extracted from bitstreams, we train simple regressors to determine the required temporal depth for accurate video classification. In doing so we show that significant redundancies in utilized data can be omitted without affecting classification accuracy. Specifically, we propose a method for cropping AVC/H.264 bitstreams to the minimum elements required to allow for the extraction of data required for video classification in compliance with the codec standard. This is to say, instead of retaining entire compressed video bitstreams and applying complex optical flow calculations prior to CNN processing, we only retain motion vector and select texture information at significantly-reduced bitrates and apply no additional processing prior to CNN ingestion. Based on three CNN architectures and two action recognition datasets, we achieve significant savings in bitrate with marginal effect on classification accuracy. Our contribution proposes a model-based selection method between multiple CNNs which increases bitrate savings further, to the point where, if up to 7% loss of accuracy can be tolerated, video classification can take place with as little as 3 kbps for the transport of the required compressed video information to the system implementing the CNN models.

In Chapter 5, we consider systems where sensors and computer vision models are distributed across communication networks to propose a method of optimizing the data utility of such system for any computer vision task. To do so, we propose a new class of mixtures of experts, where expert utility is biased by design. Our approach postulates that the minimum acceptable amount of data allowing for highly-accurate results can vary for different input space partitions. Therefore, we consider mixtures where experts require different amounts of data, and train a sparse gating function to divide the input space for each expert. By appropriate hyperparameter selection, our approach is able to bias mixtures of experts towards selecting specific experts over others. By doing so, pre-specified constraints on data transfer between the visual sensing and neural network processing parts of the system can be met while maintaining the best performance possible. To show the relation between data availability and performance, we evaluate biased mixtures on a range of well-investigated applications, namely: (i) single shot detection, (ii) realtime video action classification, and (iii) image super resolution. Our validation detailed in this chapter shows that, for all tested applications, biased mixtures significantly outperform individual experts optimized to meet the same constraints on available data.

We finally note that, implementations of our work can be adopted in commercial systems to reduce the cost of running computer vision models, and make them more prevalent in applications where processing is done remotely and computation and communication resources are scarce.

2.8 Research Outcomes

The work completed during this PhD has resulted in seven conference publications and two journal publications. There is also a conference paper to be submitted to CVPR, and another journal paper submitted to TCSVT which is under review. We also note that we only present a part of our PhD work in this thesis, and more is included in our papers. The following are our publication in **IEEE Conferences**:

1. Alhabib Abbas and Yiannis Andreopoulos, "Biased Mixtures Of Experts: Enabling Computer Vision Inference Under Data Transfer Limitations", submitted to CVPR 2020.

2. Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtslatze, "Graph-based Spatial-temporal Feature Learning for Neuromorphic Vision Sensing", in IEEE International Conference on Computer Vision (ICCV), 2019.

3. Abbas, Alhabib, Aaron Chadha, Yiannis Andreopoulos, and Mohammad Jubrani. Rate-Accuracy Trade-Off In Video Classification With Deep Convolutional Neural Networks. in IEEE International Conference on Image Processing (ICIP), 2018.

4. Aaron Chadha, Alhabib Abbas, and Yiannis Andreopoulos. Compressed do-

main video classification with deep neural networks: "There's way too much information to decode the matrix", in IEEE International Conference on Image Processing (ICIP), 2017.

5. Alhabib Abbas, Nikos Deligiannis, Yiannis Andreopoulos, and Mohammad Jubran. Vectors of Locally Aggregated Centers for Compact Video Representation in IEEE International Conference on Multimedia and Expo (ICME), 2015.

6. Aaron Chadha, Yin Bi, Alhabib Abbas, Yiannis Andreopoulos. Neuromorphic Vision Sensing For CNN-based Action Recognition, International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019.

7. Aaron Chadha, Yin Bi, Alhabib Abbas, Yiannis Andreopoulos. Neuromorphic Vision Sensing for CNN-based Action Recognition submitted to IEEE International Conference on Acoustics and Signals Processing (ICASSP), 2019.

And the following were published in **IEEE journals**:

1. Mohammad Jubran, Alhabib Abbas, Aaron Chadha, Yiannis Andreopoulos. Extension of: Rate-Accuracy Trade-Off In Video Classification With Deep Convolutional Neural Networks in IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 2018.

2. Aaron Chadha, Alhabib Abbas, Yiannis Andreopoulos. Video Classification With CNNs: Using The Codec As A Spatio-Temporal Activity Sensor in IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 2017.

3. Mohammad Jubran, Alhabib Abbas, and Yiannis Andreopoulos. "Sequence-Level Reference Frames In Inter-Frame Video Coding" in IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 2019, Submitted and is under revision.

Chapter 3

Compressed Bitstream Action Classification With CNNs

This chapter details our work on compressed video classification, and describes our solution to compressed domain action classification, where action are predicted directly from codec motion compensation parameters. Sections are organized as follows: Section 3.1 summarizes relevant aspects of video coding in the AVC/H.264 compression standard. Section 3.2 presents the proposed optical flow approximation method, which estimates optical flow directly from codec motion compensation parameters. Sections 3.3 and 3.4 details our classification model and presents our experimental results on video classification via sparse optical representations. Finally, Section 3.5 concludes this chapter. We also note that some of the details presented in this chapter are also relevant to Chapter 4, specifically with regards to the qualifying background in Sections 3.1 and 3.2 .

3.1 Video Coding And Motion Compensation

Video compression standards like AVC/H.264, HEVC, as well as open-source video codecs like Theora and Google VP8/VP9, define video bitstream formats where frames are divided into pixel blocks to be separately decoded in some specified order [54, 55, 69] . Said blocks constitute the building blocks for frame reconstruction, and form the basis for two frame prediction paradigms: inter-frame and intra-frame prediction. In general, inter-predicted blocks are reconstructed with prior knowledge of similar blocks that exist in neighboring frames, and as such, motion compensation parameters that point to the location of predictor blocks correlate with the underlying motion flow of video sequences. All current standards also define multiple options for blocks to allow for coarser or finer representations of frames, where finer representations allow for more precise block localisation and prediction. Motion compensation parameters constitute the basis on which we design our optical flow estimator.

3.1.1 Motion Estimation And Intra-Frame Prediction

Under the AVC/H.264 and HEVC standards, macroblocks are inter-predicted whenever it is possible to infer textures of a frame from previously decoded frames[54, 55]. That is, for every inter-predicted macroblock, *motion vectors* are encoded by searching local patches of previously decoded frames for macroblocks that similar textures to the macroblock being encoded. In instances where motion estimation cannot be exploited (i.e., when neighboring frames do not share any semantic features), intra-prediction is used to eliminate spatial redundancies wherever possible. Intra-prediction attempts to estimate local textures by extrapolating the textures of adjacent macroblocks. The inclusion of intra-predicted macroblocks is particularly useful in flat backgrounds where spatial redundancies often exist. Once the motion compensation parameters are decided, the difference between the macroblock predictions and the actual values that the block should assume is then calculated as additive error, and is separately encoded via entropy coding. The extent to which macroblocks are distorted is conditioned on the bitrate allowed to encode the bitstream, or on a quality parameter specified prior to encoding.

Motion compensation is preformed by partitioning frames into blocks of pixels which are subsequently used in cross-prediction (i.e., predicting on block from another). To preform texture search at a finer scale, macroblocks are searched for up to a half-pixel accuracy (i.e., frames were upscaled via interpolation to twice their size before the search was performed, which in effect produces "intermediate" pixels that allow for referencing up to a half pixel accuracy in with respect to the original frame). In many video coding scenarios, it is common that: (a) there are apparent correlations between the directions of motion vectors and the direction of movement that can be inferred from frames. For example, for the subjects appearing to be sitting at a table stationarily, motion vectors are not used, while for subjects who appear to be walking, motion vectors are likely to point to the opposite direction of their movement. This correlation is important, and will be of relevance to later parts of this work.

It is also common and natural that some motion vectors do not appear to correspond to movement. For example, vectors on the surface of an object may indicate intra-predicted matches to those specific macroblocks, and seem to be pointing towards neighboring surface textures (while the surface of the object is static). These types of motion vectors can be construed as noisy vectors, and they often occur in local regions of frames where there are repeating patterns or where there are no distinct object features in the frame (in which case such macroblocks would be more easily inferred from neighboring textures). We will also see how noise in motion vectors is correlated with coding bitrates in Chapter 4.

Frame reconstruction commonly suffers least for static areas of frames, and clear artifacts appear especially around moving regions of frames. In AVC/H.264 and HEVC, the degree of acceptable distortion is quantified and controlled via the quality parameter, or via a bitrate control parameter termed the Constant Rate Factor (CRF). Specifically, higher CRF settings and lower QP settings enforce better quality, which in turn allows for more information to be captured in motion vectors and error residuals. The principles we note above in (a) and (b) apply to older codecs (e.g., MPEG) and newer codecs alike (e.g., AVC/H.264, VP8, VP9). As such, any proposition that leverages typical outcomes of motion compensation is likely to be portable across all current standards.

3.2 Inferring Optical Flow From Macroblocks

We draw from the *de facto* approach outlined by Coimbra and Davies [201] and Kantorov *et. al* [189] for the estimation of a coarse optical flow to describe motion fields in video. Specifically, we calculate inherently sparse optical flow representations from AVC/H.264 bitstreams as the coordinate differences between source and target macroblocks, and do so while maintaining the following precepts:

1. Optical flow is estimated from bitstreams encoded using a specified quality parameter setting (i.e., no rate control is used).

2. Whenever possible, macroblocks are encoded as 8×8 pixel blocks to estimate finer motion compensation parameters.

3. Frames with no inter-predicted motion vectors (i.e., I-type macroblocks) are omitted, and replaced with frames interpolated from their immediate temporal neighbors.

4. To reduce approximation noise of inter-prediction, bi-directional macroblocks are omitted (i.e., B-frames).

5. Wherever there are "gaps" in the motion vector maps, vectors are interpolated spatially to fill a grid that supposes that all macroblocks are encoded to be 8×8 in pixels. This is performed to remove discontinuities in the motion vector map, and to generate smoother representations of motion closer to those of dense optical flow estimators.

Notable from the above, there is no point in which frames are reconstructed, and the computational advantages of our method emanate from discarding the requirement to decode video data. Moreover, we perform almost no pre-processing to estimate our optical flow relative to dense optical flow estimators such as the ones described in [202][203]. As we demonstrate experimentally in Section 3.3.5 these modifications procure significant improvements to processing speed, making our method more suitable to real-time applications.

In order to extract macroblock motion information from a compressed video bitstream, we use `avlib`, which supports most MPEG/ITU-T standards used in practice [204]. Specifically, we make use of the `AVMotion Vector` structure. When `avlib` attempts to read the compressed bit-stream of a video frame, our optical flow estimator extracts the motion compensation parameters and places them

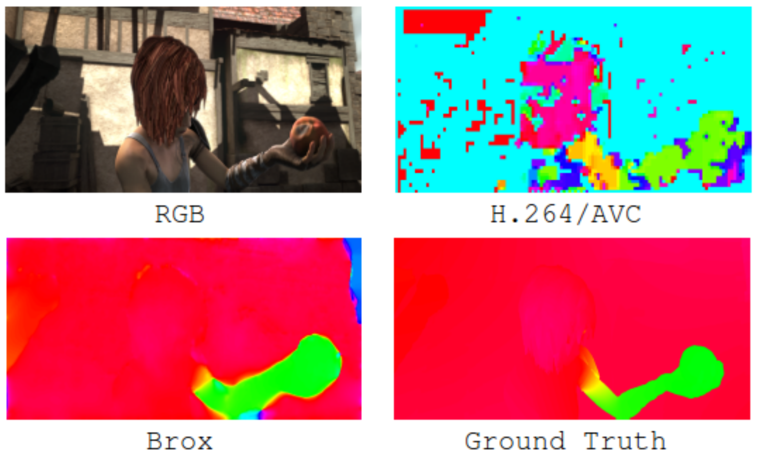

Figure 3.1: Macro-block motion vectors are derived by temporal block matching and can be interpreted as approximations of the underlying optical flow. From top-left to bottom-right: (1) RGB: frame as seen by the viewer (2) Brox: A computation-heavy optical flow approximation (3) H.264/AVC: Macro-block motion vector information rendered on a frame (4) Ground Truth: Real optical flow as generated .

in the `AVMotionVector` structure. The horizontal and vertical coordinates of the macroblocks of each frame within this structure are written in a 16-bit integer binary format to disk in order to be used by the proposed 3D deep CNN. By limiting our method to solely using the motion compensation parameters to estimate the optical flow, we achieve the speed gains reported in Section 3.3.5.

As shown in Figure-3.1, such motion vectors can be interpreted as noisy approximations of the underlying motion. The quality of macroblock based motion estimation is thus correlated with the size of macroblocks, the video resolution, and the utilized motion source search parameters.

3.3 Proposed CNN For Action Classification

In this section we describe the proposed framework for training a 3-Dimensional CNN using macroblock motion vectors. It is important to note that, for *both* training and testing with the proposed approach, *no decoding* of any video to its pixel-domain representation is performed.

3.3.1 Network Input

Using the optical flow estimation paradigm described above, we extract and retain only motion vectors extracted from uni-directional inter-predicted macroblocks, (i.e. P-type macroblocks). Empirically, we found that training on both uni-directional and bi-directional motion vectors incurs substantial increase in complexity with marginal improvements in classification accuracy. Uni-directional macroblocks are processed exclusively to reduce the complexity of the network's forward-pass during inference. To test the performance of our architecture, we use the standard action classification datasets UCF-101 [184] and HMDB-51[205]. Both are datasets composed of annotated videos with a spatial of 320×240 pixels. For a frame comprising of P-type macroblocks, a block size of 8×8 pixels results in a motion vector field of dimensions $40 \times 30 \times 2$, where $W \times H$ is the motion vector spatial resolution and the number of channels $K = 2$ is representative of the δx and δy motion vector components.

In order to compensate for the low spatial resolution $W \times H$, we take a long temporal extent of motion vectors over $T > 100$ consecutive P frames. This is contrary to recent work using high-resolution optical flow [17, 183], which typically ingest only a few frames per input (typically around 10). This is because, even with the latest GPU hardware, a long temporal extent cannot be processed without sacrificing the spatial resolution of the optical flow [17, 183]. On the other hand, given that our MB motion vector input is inherently low-resolution, it is amenable to a longer temporal extent, which is more likely to include the entirety of relevant action that is essential for the correct classification of the video. For example, we have found that the accuracy increases greatly for UCF-101 evaluated on our 3D CNN when moving from 10 to 100 frames, but eventually plateaus when T becomes sufficiently large such that the input extends to almost all P-type frames of the majority of video files of the dataset. Therefore, we fix the temporal extent T to 160, which is roughly the average number of P-frames per video in UCF-101.

3.3.2 Network Architecture

Our CNN architecture is illustrated in Figure-3.2 All convolutions and pooling are spatiotemporal in their extent. 3D pooling is performed over a $2 \times 2 \times 2$ window with spatiotemporal stride of 2. The first two convolutional layers use 3D filters of size $3 \times 3 \times 3$ to learn spatiotemporal features. With a $24 \times 24 \times 2 \times 160$ motion vector input, the third convolutional layer receives input of size $6 \times 6 \times 2 \times 10$. Therefore, we set the filter size of the third, fourth and fifth convolutional layers to $2 \times 2 \times 2$, as this is sufficiently large to encompass the spatial extent of the input over the three layers whilst minimizing the number of parameters. In order to maintain efficiency when training/evaluating, we also use a temporal stride of 2 in the first and second convolutional layers to quickly downsize the motion vector input; in all other cases we set the stride to 1 for convolutional layers. All convolutional layers and the FC6 and FC7 layers use the parametric RELU activation function [206].

It is important to note that our network has substantially less parameters and activations compared to other architectures using optical flow. In particular, our 3D CNN stores 29.4 million weights. For comparison, ClarifaiNet [207] and similar configurations that are commonly used for optical-flow based classification [17, 190] require roughly 100 million parameters.

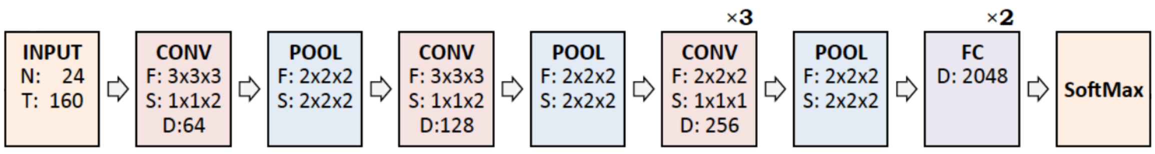

Figure 3.2: Our proposed CNN architecture: the red, blue and purple blocks represent convolutional, pooling and fully-connected layers, where: *F* is the filter size for the convolutional layers, *S* is the filter/window stride, and *D* is the depth of filters for the convolutional and fully-connected layers.

3.3.3 Training

We train the network using stochastic gradient descent with momentum set to 0.9. The initialization of He *et al.* [206] is extended to 3D and the network weights are initialized from a normal distribution with variance inversely proportional to the fan-in of the filter inputs. Mini-batches of size 64 are generated by randomly selecting 64 training videos. From each of these training videos, we choose a random index from which to start extracting the P-frame MB motion vectors. From this position, we simply loop over the P-type MBs in temporal order until we extract motion vectors over T consecutive P frames. This addresses the issue of videos having less than T total P frames, e.g., cases where the video is only a few seconds long. For UCF-101, we train from scratch; the learning rate is initially set to 10^{-2} and is decreased by a factor of 0.1 every 30k iterations. The training is completed after 70k iterations. Conversely, for HMDB-51, we compensate for the small training split by initializing the network with pre-trained weights from UCF-101 (split 1). The learning rate is initialized at 10^{-3} and decayed by a factor of 0.1 every 15k iterations, for 30k iterations.

To minimize chances of overfitting due to the low spatial resolution of these motion vector frames and the small size of the training split for both UCF-101 and HMDB-51, we supplement the training with heavy data augmentation. To this end, we concatenate the motion vectors into a single $W \times H \times 2T$ volume and apply the following steps (where T is doubled to account for two horizontal and vertical displacement channels); *(i)* a multi-scale random cropping to fixed size $N_\text{c} \times N_\text{c} \times 2T$ from this volume, by randomly selecting a value for N_c from $N \times c$ with $c \in \{0.5, 0.667, 0.833, 1.0\}$; as such, the cropped volume is randomly flipped and spatially resized to $N \times N \times 2T$; *(ii)* zero-centering the volume by subtracting the mean motion vector value from each motion vector field $\mathbf{\Phi}$, in order to remove possible bias; the δx and δy motion vector components can now be split into separate channels, thus generating our 4D network input $\hat{\mathbf{\Phi}}$. During training, we additionally regularize the network by using dropout ratio of 0.8 on the FC6 and FC7 layers together with weight decay of 0.005.

3.3.4 Testing

During testing, per video, we generate 10 random volumes of temporal size T from which to test on. Per volume, we use the standard 10-crop testing [71], cropping the four corners and the center of the image to size $N \times N \times 2 \times T$ and considering both horizontally flipped and unflipped versions. As such, we average the scores over the 10 crops and 10 volumes to produce a single score for the video.

3.3.5 Experimental Results

In order to examine the accuracy and extraction time of our approach versus decoding and optical flow estimation, we perform a comparison to the Brox [208] and FlowNet2 [209] optical flow estimations that were respectively used by Simonyan *et al.* [17] and Brox *et al.* [209]. Since the CNN architectures downsample and quantize the optical flow before ingestion [17], we measure the end-point error (EPE) of the optical flow estimations at the resolution and quantization settings used by the CNN. All runtime results were measured using an Amazon EC2 instance running on a quadcore Intel's Xeon E2686 V4 (2.3 GHz, single-thread execution). Table 3.1 presents the results using a synthetic sequence for which the ground truth motion flow is also available. At the same time, the proposed approach is more than 600 times faster than FlowNet2, as it does not decode the video to the pixel domain and does not carry out any optical flow calculations. In this regard, at current AWS spot pricing[1], GPU instances require more than 9 times the cost of CPU instances, which leads to more than 5400 times lower cost under a cloud-based deployment. million minutes of video with the state-of-the-art, our approach can process more than 8.6 billion minutes of video.

3.4 Using Codecs As Spatio-Temporal Sensors

We further extend our method by employing selectively-decoded MB texture information using the extracted MVs as activity indicators. We do this by decoding one frame every X frames, with X set to inf, indicating that only the first frame of the video is decoded. In between fully-decoded frames, "rendered" frames can be produced at frame interval R, with $1 \leq R \leq X$. Each rendered frame is initialized as a copy of the immediately preceding fully-decoded frame. Texture information at active macroblock positions is decoded and replaces the initialized values in the corresponding locations in the rendered frame. Two examples of this process are

[1]AWS EC2 spot pricing, (m3.large vs. p2.xlarge N. Virginia, Feb. 2016)

shown in Figure 3.4. We consider the area within a macroblock to be active when the corresponding MV information exceeds a specified threshold A, where $0 \leq A$. Figure 3.3 shows a grayscale activity map derived from the MVs of Figure 3.4 (b).

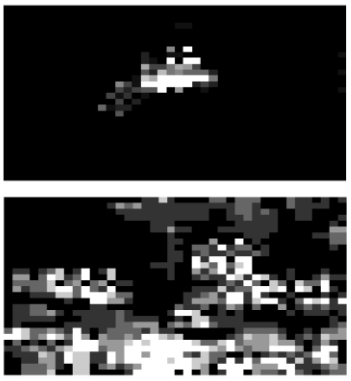

Figure 3.3: Activity regions inferred from calculated motion vectors.

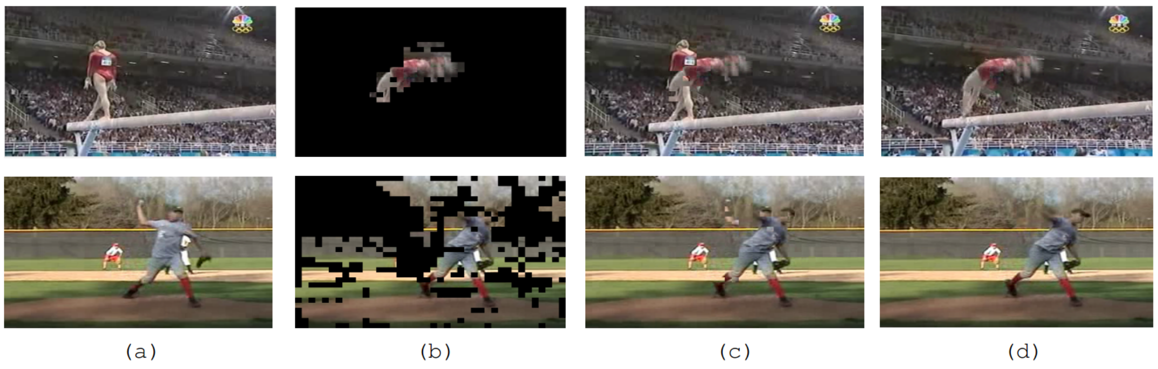

Figure 3.4: Sample from a selectively decoded sequence, with: (a) initial RGB frames, (b) selectively decoded active regions, (c) decoded regions superimposed on inital frames, and (d) ground-truth frames with full motion compensation.

To achieve said block-wise selective MB texture decoding, we inspect the motion vectors and write decoded data wherever the conditions specified by $\{X,R,A\}$ are met. Evidently, by increasing the values for $\{X,R,A\}$ we can decrease the frequency of full decoding and selective macroblock texture decoding to achieve any extraction speed desired within a practical application context.

3.4.1 Testing

During testing, per video, we generate 5 random volumes of temporal size T from which to evaluate on the temporal stream. Per volume, we crop the four corners and the center of the image to size $N_\mathrm{T} \times N_\mathrm{T} \times 2 \times T$ and consider both horizontally flipped and unflipped versions. Due to the low resolution and short duration of the HMDB-51 and UCF-101 videos, we note that taking extra crops and volumes is often redundant, as the spatial resolution of the P-frames is low and the temporal extent T of the input is large enough that it encompasses the entire video duration. However, our approach is better suited to videos “in the wild" and we can afford the extra redundancy due to the low complexity of our 3D CNN. As a result, rather than computing an average score over all extracted volumes, we simply take the maximum score, in order to generate our prediction for the video.

As the VGG-16 architecture has roughly 6 times the number of weights and activations of our proposed 3D CNN in the temporal stream, we evaluate on the spatial stream by extracting 5 frames from the set per video albeit with a single center crop (and its horizontal flip) of size $N_\mathrm{S} \times N_\mathrm{S} \times 3$. To generate our prediction, we again compute the maximum score over all extracted frames.

3.4.2 Structural Similarity

For the experiments with the proposed approach, we chose values for the decoding interval X that correspond to the settings used in our video classification tests. In addition, for both cases, we set the rendering frame interval to $R = 10$ and threshold $A = 0$. Under these settings, the EPE of the proposed approach remains low-enough to indicate high correlation with the ground-truth and optical-flow based methods.

In terms of visual quality of the selective decoding and rendering approach of Section 3.2, we measure the average structural similarity index metric (SSIM) [25] using the fully-decoded video sequences as reference. By using 100 video sequences from UCF-101, and for a range of values of the decoding interval X (all other settings remain the same as for Table 3.1), the measured drop in visual quality is reported in Figure 3.5. Evidently, the quality of rendered frames remains high enough to allow for all SSIM values to remain above 0.85. This agrees with the visual similarity we observed in typical examples such as the ones of Figure 3.4(c)+(d). We can fuse the two streams together by simply averaging their maximum scores. Finally, in order for our spatial stream to have temporal correspon-

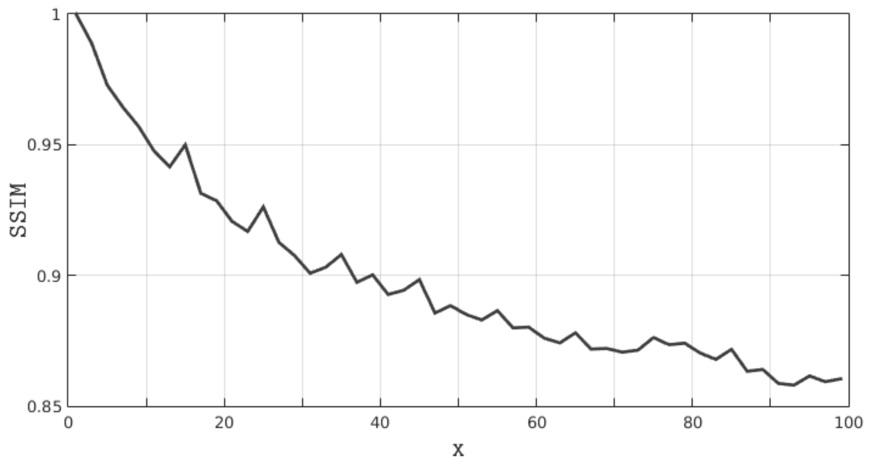

Figure 3.5: Structural similarity index of decoding with varying intervals X.

dence with our temporal stream, we pass the starting index of each P-frame volume.

3.4.3 Experimental Results

Table 3.1 presents results from our MB motion vector extraction against ground truths, Brox [208], and FlowNet2 [209] optical flow estimations that were respectively used by [17] and [209]. All runtime results were measured using an Amazon EC2 instance running on a quadcore Intel's Xeon E2686 V4 (2.3 GHz). Since the CNN architecture downsamples and quantizes the optical flow before using it [17], we measure the end-point error (EPE) of the optical flow estimations at the resolution and quantization settings used by the CNN. Under these settings, the EPE of the proposed approach remains low-enough to indicate high correlation with the ground-truth and optical-flow based methods.

3.4.4 Datasets

Evaluation is performed on two standard action recognition datasets, UCF-101 [184] and HMDB-51 [205]. UCF-101 is a popular action recognition dataset, comprising 13K videos from 101 action categories. Each video is: approximately 10 seconds in duration, 320×240 pixels per frame, at 25 frames per second (fps). HMDB-51 is a considerably smaller dataset, comprising only 7K videos from 51 action categories, with the same spatial resolution as UCF-101, and at 30 fps.

Input	Decoding	Flow Estimation	
	Runtime	Runtime	EPE
Proposed, $X = 50$	384.6 (CPU)	18602 (CPU)	15.26
Brox	120.0 (CPU)	0.16 (GPU)	6.32
FlowNet2	120.0 (CPU)	8.13 (GPU)	3.14

Table 3.1: Decoding and motion field estimation accuracy and runtime (fps) results for the proposed approach, Brox [208] and FlowNet2 [209]. EPE: end-point error; SSIM: structural similarity index metric.

Framework	Input	Complexity	Accuracy (%)	
	Size	#A, #W ($\times 10^6$)	UCF	HMDB
Proposed 3D CNN	$24^2 \times 2 \times 160$	4.0, 29.4	77.5	49.5
TSCNN-Brox	$224^2 \times 20$	2.0, 90.6	83.7	54.6
LTC-Brox	$58^2 \times 2 \times 100$	42.1, 12.2	82.6	56.7
LTC-Mpegflow	$58^2 \times 2 \times 60$	25.3, 10.6	63.8	–

Table 3.2: Comparison with state-of-the-art flow based networks. “Proposed 3D CNN” refers to our temporal stream that ingests MB motion vectors. Complexity is reported with respect to millions of activations and weights (#A, #W), summed over conv, pool and FC layers in the utilized deep CNN of each approach.

3.4.5 Evaluation Protocol and Results

For each dataset we follow the testing protocol of Section 3.4.1 and compute the average accuracy over the three training/test splits provided. Each UCF-101 training split consists of approximately 9.5K videos, whereas each HMDB training split has 3.7K videos. It is evident from Table 3.2 that our proposal outperforms the RGB-based SSCNN [17], LTC-Mpegflow [18] and SFCNN [183]. At the same time, our proposal is outperformed by SSCNN-Brox and LTC-Brox (both using highly-complex optical flow), as well as the RGB-based C3D [187], by up to 10 percentile points. Importantly, our approach allows for more than two-fold reduction in the number of activations and weights against competing methods, and we mainly attribute the gap in performance to the short duration of the videos in the datasets, and the low-resolution nature of used content.

3.5 Concluding Remarks

In this chapter we showed how codec motion vectors can stand in as cost-effective representations of motion flow. We also proposed a new compressed-domain model for video classification based on deep learning methods, and discussed its mode of operation and performance. We showed that our method achieves accuracy comparative to the state-of-the-art, with speed three orders of magnitude higher than that of models which require the estimation of dense optical flow prior to inference. Our proposal in this chapter can be interpreted as an input dimensionality reduction method that leverages the correlation between block-based texture search methods and motion flow that underlies video sequences. In the next chapter, we use our findings on compressed domain video classification to extend our study to data utility optimization, and do so for the express purpose of video classification.

Chapter 4

Rate-Accuracy Tradeoff For Compressed Action Classification

This chapter details our study on the trade offs between rate control in video coding and video classification accuracy, and describes our proposed bitrate optimization model for video classification. Specifically, we demonstrate how classification of video data can be reliably achieved by models that exists on remote machines without relaying redundant information from sensors. To achieve the latter, we start by exploring rate-accuracy trade-offs in CNN-based classification, and we summarize the contributions detailed in this chapter in the following:

1. We study the effect of varying encoding parameters on state-of-the-art CNN-based video classifiers. Unlike conventional rate-distortion curves, we show that, without any optimization, rate-accuracy is not monotonic for CNN-based classification.

2. In order to optimize the trade off between bitrate and classification accuracy, we propose a mechanism to select amongst 2D/3D temporal CNN and spatial CNN classifiers that have varied input volume requirements, and we achieve this with bitstreams that comply with standardized video codec formats.

3. We study and compare the efficacy of our method on action recognition based on AVC/H.264 and HEVC compressed video, which represent two of the most commonly-used video coding standards.

Sections are organized as follows: Section 5.1.1 describes how video bitstreams are reduced through selective cropping. In Section 4.2, we describe and formulate the optimized classifier selection process. Section 4.4 evaluates the performance of the proposed classifiers using different coding settings and illustrates the rate gains made possible through our classifier selection method. Finally, Section 4.5 concludes the chapter.

4.1 Cropped Video Bitstreams

We base our reduced-bitstream encoder on the JM reference software of AVC/H.264 [69] and the HM reference software of HEVC [54]. Our modifications to the reference encoders are designed such that the bitrate of the compressed bitstream is kept at a minimum while preserving the information needed to classify videos. Namely, the compressed bitstream should exclusively hold: *(i)* key texture components corresponding to rapidly-changing input content; *(ii)* inter-frame predicted macroblocks and their motion compensation parameters; *(iii)* control signals and headers needed to comply with its corresponding standard.

4.1.1 Inferring Optical Flow From Cropped Bitstreams

Before applying inter-frame prediction, AVC/H.264 pictures are split into 16×16 pixel macroblocks (MB) to represent luminance and chrominance samples, with the chrominance samples further split into 8×8 chroma blocks for the widely used 4:2:0 chroma sampling. Macroblocks are the core of the coding layer and form the basis for adaptive inter and intra predictions. Each of the inter-predicted macroblocks is then encoded using blocks from the set $\{16 \times 16, 16 \times 8, 8 \times 16, 8 \times 8\}$ [54, 210]. The HEVC standard takes on a more adaptive approach and introduces a Coding Tree Unit (CTU) which consists of luma and chroma Coding Tree Blocks (CTB). The size of each luma CTB is drawn from the set $\{16 \times 16, 32 \times 32, 64 \times 64\}$ where larger size blocks result in better compression efficiency. Iterative partitioning is then applied to divide CTBs into smaller Coding Blocks (CB) resulting in a tree-like structure [211]. The minimum allowed CB size is also specified, this serves as a parameter to control the granularity of the tree structure produced [56].

In both standards, blocks are predicted via translational motion vectors (MVs) that represent the displacement from matching blocks in previous or subsequent reference frames. Increasing the number of small-size blocks increases the granularity of the MV grid at the expense of lower coding efficiency. These MVs represent the temporal activity and have been shown to be highly correlated with optical flow estimates [121]. If the area covered by the MB is static, the MB is "skipped" and is not encoded. The resulting prediction residual from temporal prediction of non-skipped MBs is encoded using transform coding. The transform coefficients are then quantized based on the quantization parameter (QP). The value of the QP per frame can be chosen from 52 values in $[0,51]$, with lower values indicating high-fidelity encoding.

4.1.2 Selective Retention of Motion and Texture Information

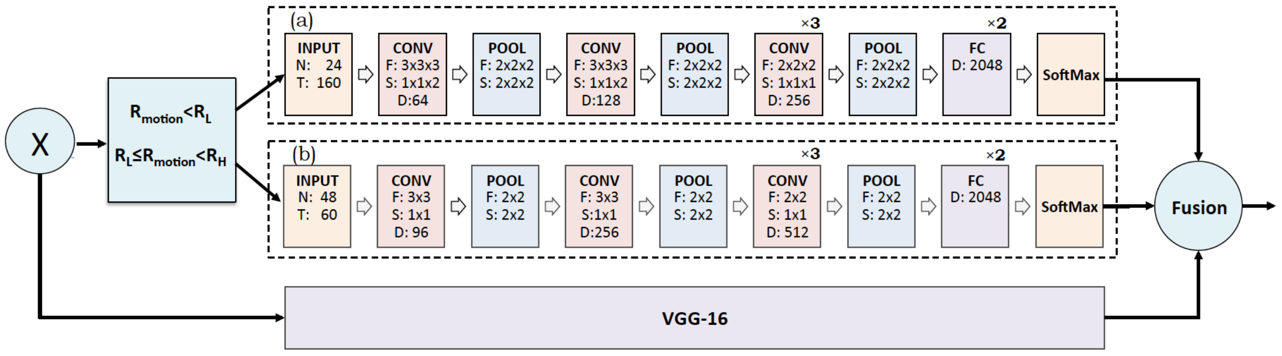

Figure 4.1: The proposed Multi-CNN classifier selection: (a) 3D temporal CNN architecture; (b) 2D temporal CNN architecture. The bottom part represents the spatial CNN (VGG-16). Parameters: N is the spatial dimensions of the input volume; T is the temporal extent expressed as the number of frames used; F is the filter size, formatted as width $\times$ height $\times$ time; S is the convolutional window stride; D is the number of filters (or number of hidden units) for the convolutional and fully-connected layers; R_{L} and R_{H} are controlling the multi-CNN selection based on the motion vector rate R_{motion}.

In our work, only select subsets of the quantized transform coefficients will be entropy encoded and then included in the cropped bitstream. This set of coefficients, along with spatial texture, is transmitted to the classifier (described in Section 4.2) to infer semantic features and classify the content of the bitstream. By doing so, the bitrate of these "cropped" subsets of coefficients, R_{cropped}, is significantly reduced in comparison to the original bitrate, R_{orig}, needed to encode the full video. In the remainder, we present our modifications, assuming that the first frame of every video sequence is encoded as an Instantaneous Decoding Refresh (IDR) and all subsequent frames in the video are encoded as P-frames.

In order to reduce the bitrate of the compressed bitstream, we employ selective retention of texture information by retaining the texture information of active regions. To implement selective writing in the AVC/H.264 JM reference software [69], we modified functions `writeCoeff4x4_CAVLC_normal()` and `write_chroma_intra_pred_mode()`. In addition, to allow for a skip symbol for all non-active areas, we modified the functions `read_coeff_4x4_CAVLC()` and `read_coeff_4x4_CAVLC_444().` Similarly, to implement selective writing in the HEVC HM reference software [56], we modified the functions `TEncSbac::codeCoeffNxN()` and `TDecSbac::parseCoeffNxN().` To simplify our tests, we retain the texture of IDR frames and skip all texture of P-frames with a single skip symbol. The introduction of these skip symbols is the only non-normative part of our entire process. All other syntax elements (including modes and motion information) are left as specified in their respective standard. With these minimal changes, standard decoders can decode our reduced bitstreams to pass to compressed video classifiers.

Finally, in order to derive a temporal activity map from P-frame MVs, we apply the following steps: *(i)* MVs are extracted from the compressed bitstream using the `read_motion_info_from_NAL_p_slice()` function for JM and `TDecEntropy::decodePUWise()` for HM; *(ii)* the extracted MVs are then mapped to a grid of 8×8 non-overlapping blocks within each frame; *(iii)* MVs are interpolated from neighboring macroblocks wherever a macroblock does not provide motion compensation parameters but two or more of its neighbors do.

4.2 Proposed Framework For Compressed-domain Classification

4.2.1 CNN Architectures

In Fig. 4.1 we illustrate the two CNNs used for the temporal MV stream, which represent the state-of-the-art in compressed-domain deep learning for action classification[121][190]. We use two architectures to study how different models behave to cropped bitstream volumes, and to demonstrate that our rate optimized CNN-based classification method is applicable with different network architectures that have been shown to preform well with codec motion vector data. The first CNN architecture we consider is the 3D CNN proposed by Chadha *et al.* [121]. As illustrated in Fig. 4.1(a), all convolutional and pooling layers are spatiotemporal in extent; this captures the motion information between consecutive motion vector frames. Crucially, the spatiotemporal features are expected to improve classification performance between similar actions. We generate a 4D motion vector input by splitting the dx and dy vector components into separate channels, thus resulting in a $W \times H \times 2 \times T$ volume. We compensate for the low resolution of the extracted motion vector frames by setting a long temporal extent T as $T_{3D} = 160$, which typically comprises the entire video duration.

The second architecture we consider is a 2D CNN, as illustrated in Fig. 4.1(b). The model design is based on ClarifaiNet [212] and only comprises 2D spatial filters; we notably reduce the size of the first filter from 7×7 to 3×3 and decrease the stride of the first two convolutional layers to 1×1. A similar architecture was also employed in recent work on fast video classification [190]. The input is generated by stacking the motion vector dx and dy components into a single $W \times H \times 2T$ volume, where the temporal depth T is set as $T_{2D} = 60$. In general, 2D CNNs are less complex to train and test with than 3D CNNs, whilst forgoing modelling any temporal dependencies. Nonetheless, their lower complexity means we can afford to use a higher input spatial resolution, which enables the 2D filters to learn more distinguishing spatial features of the MV data.

Finally, concerning spatial processing of RGB texture, we use the well-established VGG-16 [188] CNN architecture to classify RGB frames and capture motion-invariant spatial features of video content. Our spatial CNN is pre-trained on ImageNet [213] and fine-tuned on the training split of UCF-101. The spatial stream ingests the decoded frames per video and the predictions made by the spatial CNN are ultimately fused with the predictions from the temporal stream to produce the final two-stream classifier decisions.

4.2.2 Training and Testing

We train both temporal stream architectures using stochastic gradient descent with momentum set to 0.9. The initialization of He *et al.* [206] is used and weights are initialized from a normal distribution. Mini-batches of size 64 are generated by randomly selecting 64 training videos per batch. The training is completed after 90k iterations. We follow the data augmentation practices utilized in recent work [121] in order to minimize overfitting for both the 2D and 3D CNN. These include a multi-scale random cropping of the input and spatial resizing to a fixed size N, followed by zero centering the motion vector field by subtracting the mean motion vector from the volume. For the 3D CNN, the fixed crop size is set to 24, whereas for the 2D CNN this is doubled to 48. In addition, we use a dropout ratio to 0.5 for the first two fully connected layers in both models. During testing, for the temporal stream we generate 10 random volumes of temporal size F from which to test on. Per volume, we use the standard 10-crop testing, cropping the four corners and the center of the image to spatial size $N \times N$ and considering both horizontally flipped and unflipped versions. As such, we average the scores over 10 crops and 10 volumes to produce a single score for the video. For the spatial stream, we use one IDR frame for each video and oversample inputs to VGG-16 by flipping and extracting crops.

4.2.3 Multi-CNN Classifier

In order to optimize the tradeoff between bitrate and classification accuracy, we leverage the differences in input requirements of the two temporal classifiers of

Fig. 4.1 and devise a Multi-CNN (MCNN) selection process. Since the number of MV frames per crop is larger for our 3D CNN versus its 2D CNN counterpart (i.e., $T_{3D} > T_{2D}$), the former requires higher bitrate per crop than the latter. On the other hand, as shown in previous studies [29], denser MV frames will benefit from the spatially-larger input of the proposed 2D CNN architecture. Since the density of inputs to the temporal stream is directly proportional to the average bitrate allocated to MVs by the codec R_{motion}, we expect the accuracy of both the 2D CNN and 3D CNN classifiers to be directly related to R_{motion}, albeit up to a limit (since noise is introduced at high rates due to the limitations of the MV block model). Moreover, the two classifiers are expected to be comparable in accuracy over a range of R_{motion} values. These hypotheses have been tested and we present the related experimentally derived results in Section 4.3.1. In summary, our investigation showed that: *(i)* the long temporal extent 3D CNN classifier is superior for lower values of R_{motion}; *(ii)* the short temporal extent 2D CNN classifier performs as well as the long temporal extent 3D CNN classifier for mid-range values of R_{motion} ; *(iii)* both temporal CNNs offer diminishing performance for high values of R_{motion}. Therefore, we introduce the pair of rate-accuracy optimization parameters $\{R_{\text{L}}, R_{\text{H}}\}$, with $R_{\text{H}} > R_{\text{L}}$, such that:

1. the 3D CNN is used for videos with $R_{\text{motion}} < R_{\text{L}}$
2. the 2D CNN is used for videos with $R_{\text{L}} \leq R_{\text{motion}} < R_{\text{H}}$
3. no temporal CNN is used when $R_{\text{motion}} \geq R_{\text{H}}$ and only the output of the spatial CNN is considered (see Fig. 4.1).

The remainder of this section is to establish a model-based approach for the optimal selection of $\{R_{\text{L}}, R_{\text{H}}\}$. While the value of R_{motion} is derived experimentally during the encoding of each video, for offline rate-accuracy optimization studies it can also be derived via rate-distortion models [214].

4.2.4 Problem Formulation and Optimization of MCNN

To make full use of the overlap of performance between classifiers, a video is passed to a lower-rate classifier only when it is likely to be classified correctly. We consider

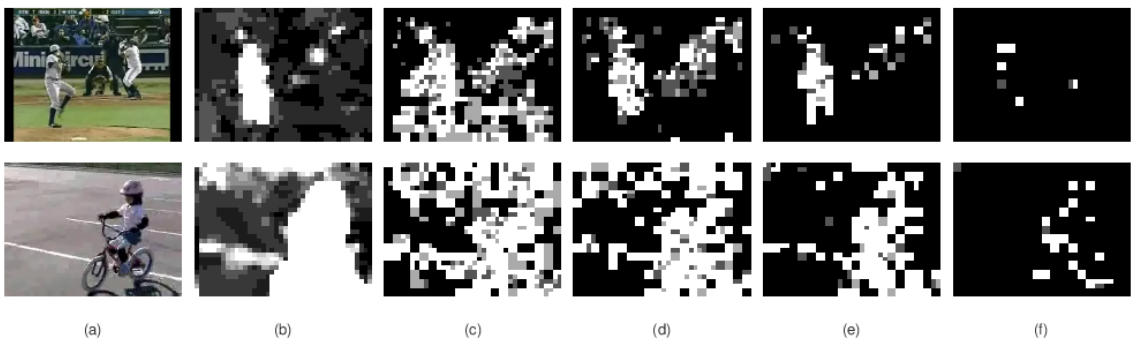

Figure 4.2: RGB frames and corresponding AVC/H.264 MV activity maps for two scenes from UCF-101; (a) RGB frames; (b) Brox optical flow; (c) Approximated flow at QP = 0; (d) Approximated flow at QP = 30; (e) Approximated flow at QP = 40; (f) Approximated flow at QP = 51. Note that sparsity increases and noise decreases with increased QP.

the problem of finding the optimum set $\{R^*_{\text{L}}, R^*_{\text{H}}\}$ that maximizes the classification accuracy, A_{mcnn}, of our proposed MCNN under a constraint on the available bitrate, $R_{\text{available}}$:

$$\{R^*_{\text{L}}, R^*_{\text{H}}\} = \underset{R_{\text{L}},R_{\text{H}}}{\text{argmax}} A_{\text{mcnn}}(R_L, R_H) \ \text{ subject to } \ R_{\text{sent}} \leq R_{\text{available}} \tag{4.1}$$

where R_{sent} is the average bitrate of all transmitted bitstreams under a selection algorithm for $\{R_{\text{L}}, R_{\text{H}}\}$. We first consider the video source probability density function $f_{\text{s}}(R_{\text{motion}})$, which characterizes the probability of occurrence of video examples with bitrate R_{motion}. We have found $f_{\text{s}}(R_{\text{motion}})$ to be well approximated by the Gamma distribution, $f_{\text{s}}(R_{\text{motion}}; \alpha, \beta)$, where α and β are the shape and rate parameters (see Section 4.3.1 and Fig. 4.3). We can then express A_{mcnn} as:

$$\begin{aligned} A_{\text{mcnn}} = A_{\text{3D}} &\int_0^{R_{\text{L}}} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \\ +A_{\text{2D}} &\int_{R_{\text{L}}}^{R_{\text{H}}} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \\ +A_{\text{SP}} &\int_{R_{\text{H}}}^{\infty} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \end{aligned} \tag{4.2}$$

where A_{3D}, A_{2D} and A_{SP} are the classification accuracies of the 3D, 2D and spatial stream classifiers respectively. In (4.2), the accuracy of each of the classifiers is assumed to be constant for the range of rates it corresponds to, and its estimate is experimentally derived from $\mathcal{V}$. This assumption holds as long as $\mathcal{V}$ is large enough and the accuracy of each classifier remains relatively flat for different values of R_{motion} within the respective integration interval of each classifier, which is found to be the case in our experiments of Section 4.4.

Since the number of bits needed to classify each video depends on which classifier is used for prediction, we first find the average bitrate required by each classifier. We define R_{3D}, R_{2D}, and R_{SP} as the average bitrate of inputs to the 3D, 2D, and spatial classifiers, respectively, and estimate each as:

$$R = \begin{cases} R_{\text{3D}} = a_{\text{3D}} R_{\text{motion}} + b_{\text{3D}} & 0 \ < \ R_{\text{motion}} < R_{\text{L}} \\ R_{\text{2D}} = a_{\text{2D}} R_{\text{motion}} + b_{\text{2D}} & R_{\text{L}} \leqslant R_{\text{motion}} < R_{\text{H}} \\ R_{\text{SP}} = I_{\text{SP}} & R_{\text{H}} \leqslant R_{\text{motion}} < \infty \end{cases} \tag{4.3}$$

where a_{3D}, b_{3D}, a_{2D}, and b_{2D} are coefficients to be estimated by applying regression on the bitrate feature R_{motion} obtained on the training set $\mathcal{V}$. Since the inputs passed to the 3D and 2D classifiers consist only of the motion vectors and some added headers to comply with the used standard, we expect the linear relations shown in (4.3) and confirm this in Section 4.3.2. For the spatial classifier, we use I_{SP}, i.e., the bitrate of the first IDR frame, to estimate R_{SP}. Note that R_{motion} is not used for R_{SP}, since the spatial classifier only uses texture information. We can now express R_{sent} as:

$$\begin{aligned} R_{\text{sent}} = & \int_0^{R_{\text{L}}} R_{\text{3D}} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \\ & + \int_{R_{\text{L}}}^{R_{\text{H}}} R_{\text{2D}} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \\ & + \int_{R_{\text{H}}}^{\infty} R_{\text{SP}} f_{\text{s}}(R_{\text{motion}}) dR_{\text{motion}} \end{aligned} \tag{4.4}$$

Based on the expectation value property of the Gamma density function $f(X;\alpha,\beta)$ [215]:

$$X f(X;\alpha,\beta) = \frac{\alpha}{\beta} f(X;\alpha+1,\beta) \tag{4.5}$$

from (4.2) and (4.4) we can rewrite A_{mcnn} and R_{sent} as:

$$\begin{aligned} A_{\text{mcnn}} &= (A_{\text{3D}} - A_{\text{2D}}) F_{\text{s}}(R_{\text{L}}; \alpha, \beta) \\ &+ (A_{\text{2D}} - A_{\text{SP}}) F_{\text{s}}(R_{\text{H}}; \alpha, \beta) + A_{\text{SP}} \end{aligned} \tag{4.6}$$

$$\begin{aligned} R_{\text{sent}} = (b_{\text{3D}} - b_{\text{2D}}) F_{\text{s}}(R_{\text{L}}; \alpha, \beta) \\ + (b_{\text{2D}} - I_{\text{SP}}) F_{\text{s}}(R_{\text{H}}; \alpha, \beta) \\ + (\alpha/\beta)(a_{\text{3D}} - a_{\text{2D}}) F_{\text{s}}(R_{\text{L}}; \alpha + 1, \beta) \\ + (\alpha/\beta)(a_{\text{2D}}) F_{s}(R_{\text{H}}; \alpha + 1, \beta) + I_{\text{SP}} \end{aligned} \tag{4.7}$$

where F_{s} is the cumulative distribution function of f_{s} and we have explicitly indicated the dependence on the parameters α and β since they affect the bitrate and accuracy contributions of the 2D and 3D CNN models. The constrained optimization problem of (4.1) can now be solved for $\{R_{\text{L}}^{*}, R_{\text{H}}^{*}\}$ via (4.6) and (4.7). We first note that (4.6) is monotonically increasing in function of R_{L} and R_{H}, since $A_{3D} > A_{2D}$ and $A_{2D} > A_{SP}$. This allows for the use numerical methods that gradually explore the parameter space of $\{R_{\text{L}}, R_{\text{H}}\}$ by setting R_{sent} in (4.7) as close as possible to $R_{\text{available}}$ and then finding the maximum values for $\{R_{\text{L}}, R_{\text{H}}\}$ that satisfy (4.7), since such values will automatically maximize (4.6).

In our experiments, amongst several alternatives, we opted for the method of Toint *et al.* [216], which finds the solution $\{R_{\text{L}}^{*}, R_{\text{H}}^{*}\}$ that maximizes (4.6) under the constraint $R_{\text{sent}} \leq R_{\text{available}}$ with the provision of sufficient exploration time. Given that this optimization process is done offline based on training data $\mathcal{V}$, this does not impose any overhead at runtime. Finally, we remark that, in case R_{motion} is not measurable at training or test time, the optimization method proposed in this section can be generalized to other features that correlate with R_{motion} (e.g. number of MVs per frame).

4.3 Validation Of Rate-Accuracy Assumptions

We validate our modelling choices described in Section 4.2.4. For brevity of exposition, all figures and results here are reported for the indicative case of AVC/H.264 with $\text{QP} = 40$.

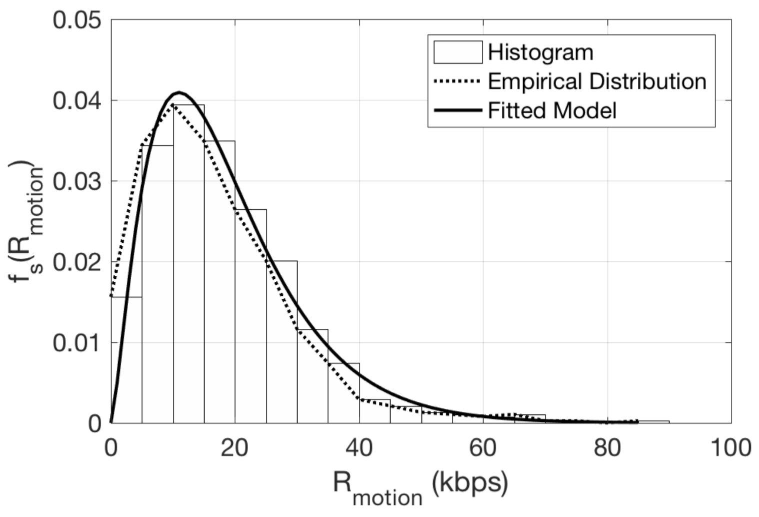

Figure 4.3: Empirically measured distribution of R_{motion} and fitted Gamma distribution with shape and scale parameters: $\alpha = 2.43$, $\beta = 0.13$.

4.3.1 Distribution of R_{motion} and Performance Overlap

In this section we compare the distribution of R_{motion} against the fitted model and verify the overlap of performance between the proposed architectures in Section 4.2. All of the UCF-101 dataset is used to produce the results shown in Fig. 4.3 and Fig. 4.4. For Fig. 4.3, the Kullback-Leibler divergence (describing the distance between the empirical and fitted Gamma distribution) was found to be 0.034. This proximity justifies our use of this distribution for characterizing the probability of occurrence of different values of R_{motion}. Concerning Fig. 4.4, the experiments show that the 3D and 2D CNN architectures perform similarly for middle-range values of R_{motion}, with the 3D-CNN outperforming the 2D-CNN for most of the lower MV bitrates. The performance of both CNNs decays for high values of R_{motion}. Hence, for the high-end range of R_{motion}, only the spatial CNN should be used (VGG-16 of Fig. 4.1).

4.3.2 Linear Model Verification for (4.3)

We selected 5% of the UCF-101 videos randomly and present the plots of R_{motion} vs. R_{3D} and R_{2D} in Fig 4.5 and Fig. 4.6 . Using the same set, we calculated the coefficient of determination R^2 to relate the experimental variance to the residual

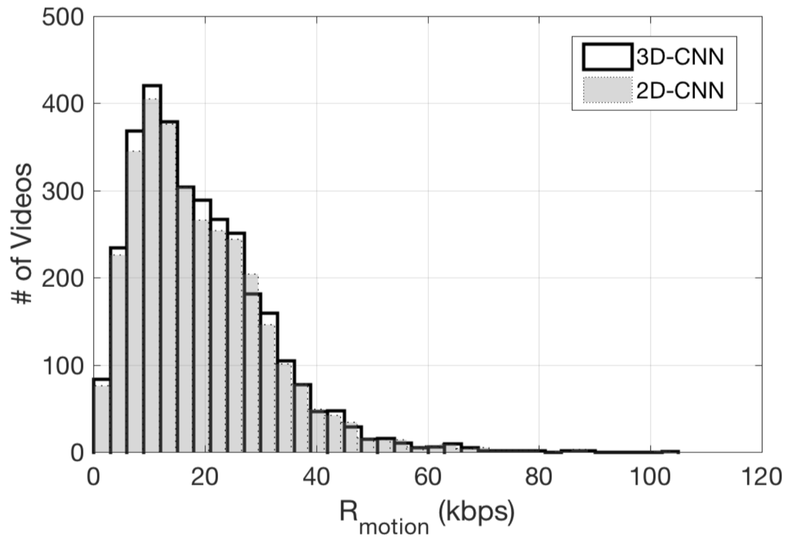

Figure 4.4: Number of videos classified correctly by each temporal CNN classifier for different values of R_{motion}.

variance of the linear model and found it to be 93% for R_{3D} and 88% for R_{2D}. Similar results have been obtained for the HMDB dataset. These results validate that the linear assumption of (4.3) is a good approximation.

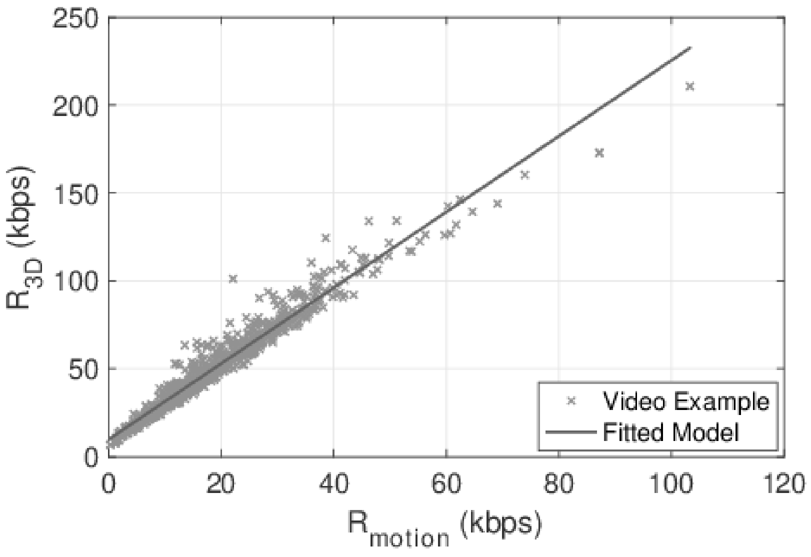

Figure 4.5: Bitrate of inputs sent to 3D architecture R_{3D} plotted against R_{motion} and fitted model of R_{3D} with linear coefficients $a_{\text{3D}} = 2.21$ and $b_{\text{3D}} = 9.04$.

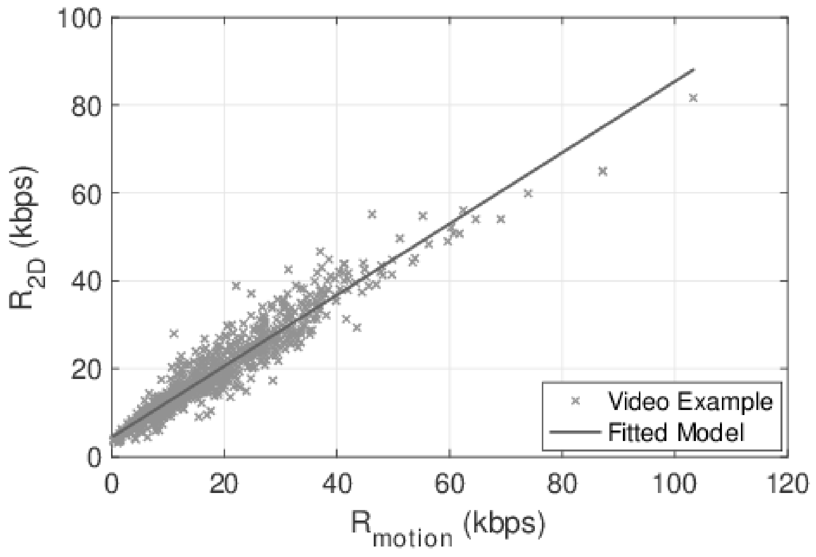

Figure 4.6: Bitrate of inputs sent to 2D architecture R_{2D} plotted against R_{motion} and fitted model of R_{2D} with linear coefficients $a_{\text{2D}} = 0.83$ and $b_{\text{2D}} = 4.27$.

4.4 Experimental Results

4.4.1 Used Datasets And Rate Saving from Cropped Bitstreams

We train and test our 2D and 3D CNN architectures on eight distinct motion vector datasets generated by varying the QP setting of AVC/H.264 and HEVC to encode UCF-101[184], while skipping texture information as described in Section 5.1.1. For all videos: the first frame is encoded as an IDR (with remaining frames inter-predicted as P-frames), the frame rate is set to 25, and we set the motion vector search range to 16 pixels. Since specifying a particular quantization parameter has a direct effect on the MVs produced by AVC/H.264 and HEVC, this gives several distinct source distributions for the classifier to be trained and tested on.

For each dataset we follow the protocol of Section 3.4.1 and compute the average accuracy over the three training/test splits provided. Each UCF-101 training split consists of approximately 9.5K videos of sequences representing 101 different subsets of apparent actions. Importantly, the classifiers we study allow for a two-fold decrease in the number of required activations and weights (which directly correlate with complexity) when compared against full-density optical flow meth-

Table 4.1: Average AVC/H.264 bitrate (kbps) of UCF-101; R_{orig} is the bitrate of the original bitstream, R_{cropped} is the bitrate after cropping and retaining texture and motion information, and R_{motion} is the MV bitrate.

				% of R_{motion} to	
QP	R_{orig}	R_{cropped}	R_{motion}	R_{orig}	R_{cropped}
0	4273.0	321.3	155.4	3.6	48.3
30	274.9	112.3	46.9	17.0	41.7
40	80.0	49.9	18.5	23.2	37.1
51	27.7	20.0	4.6	16.7	23.1

Table 4.2: Average HEVC bitrate (kbps) of UCF-101; R_{orig} is the bitrate of the original bitstream, R_{cropped} is the bitrate after cropping and retaining texture and motion information, and R_{motion} is the MV bitrate.

				% of R_{motion} to	
QP	R_{orig}	R_{cropped}	R_{motion}	R_{orig}	R_{cropped}
0	3065.2	204.9	39.9	1.3	19.1
30	157.7	58.8	12.0	7.6	20.6
40	40.2	26.7	4.9	2.5	12.25
51	10.9	9.8	0.8	7.3	8.1

ods, and we attribute the gap in performance between the two mainly to the low temporal and spatial depths of content comprising used UCF-101[184].

4.4.2 Rate-Accuracy Results

As the quality of predictions made by CNN models is strongly tied to the properties of source distributions (e.g. cross-class variance, noise), we expect that varying the rate should affect the accuracy of our classifier accordingly. Since the QP values control the video rate, we first show visual examples of the effect of QP on the quality of approximated sparse optical flow in Fig. 4.2. The best approximations

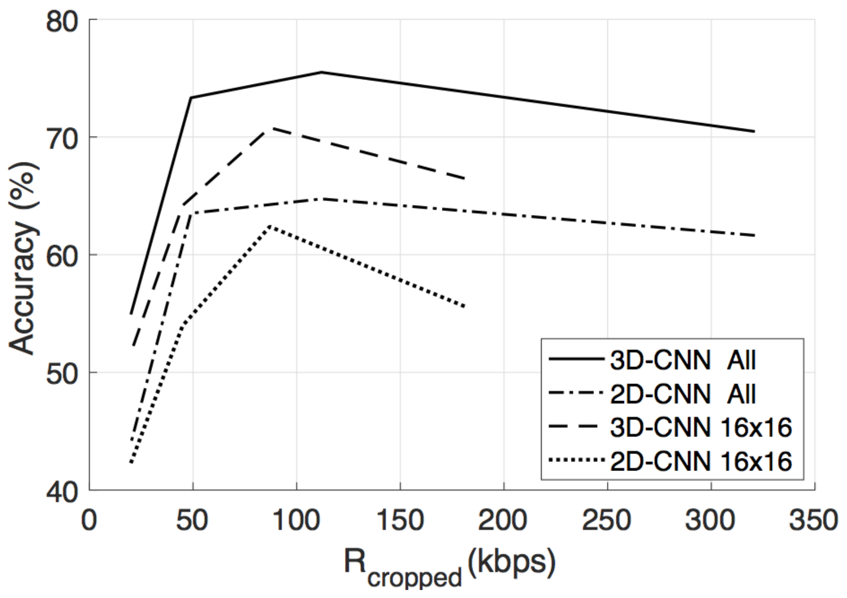

Figure 4.7: Rate-accuracy after cropped AVC/H.264 bitstreams are passed to the 2D and 3D classifiers. Each point for every curve corresponds to a different QP setting during encoding, with "16×16" indicating restriction to 16×16 blocks (no MB subblocks) and "All" indicating the use of all MB partitions.

appear to be for QP values in the region of 30 to 40. To assess the rate savings and classification accuracy of our proposal when varying QP values, in Table 4.1 and Table 4.2 we compare the original bitrate, R_{orig}, with the bitrate of the cropped bitstreams, R_{cropped}, and the rate of retained motion vectors, R_{motion}. The results show that streaming cropped bitstreams allows for 28% to 92% reduction in bitrate for AVC/H.264, and 11% to 94% for HEVC. The related classification accuracy results are presented in Fig. 4.7 and Fig. 4.8. As indicated by the visual examples of Fig. 4.2, the utilized CNNs indeed achieve their best accuracies at QP values of 30 to 40.

Importantly, we observe that rate-accuracy curves are not monotonic (i.e., accuracy decreases for very low or very high QP values). We expect sparser motion vectors (e.g., MVs produced by setting $\text{QP} = 51$ where the rate allocated to motion vectors is the lowest) to make certain classes with high motion similarity particulary harder to classify and easier to confuse with each other. On the other hand, as

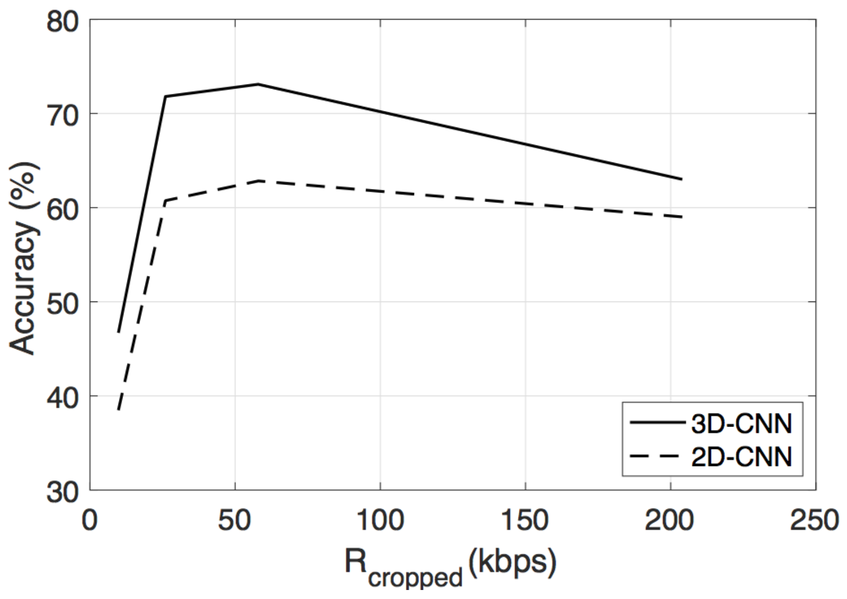

Figure 4.8: Rate-accuracy after cropped HEVC bitstreams are passed to the 2D and 3D temporal CNNs. Each point for every curve corresponds to a different QP setting during encoding, with encoder parameter CBT Depth $= 2$.

shown by Fig. 4.2, setting QP < 30 also has a detrimental effect on accuracy, since the derived MVs become significantly more noisy due to the inadequacy of the simple translational block model of AVC/H.264 and HEVC to smoothly approximate the optical flow field since such block models are optimized for rate control and not optical flow estimation [121, 208].

To cross validate with an external benchmark, Fig. 4.9 shows the average End Point Error (aEPE) between MV frames and a dense optical flow ground truth approximated using the method proposed by Brox *et al.* [186]. The resulting curves show that, for both video coders, the minimum aEPE value against dense optical flow is in the QP range of 30 to 40. We also note that the best performance occurs at a lower rate for HEVC compared to AVC/H.264, which is due to the enhanced coding efficiency and improved inter-frame macroblock search of the HEVC standard. This is also reflected in Fig. 4.9, where the aEPE of HEVC is lower than that of AVC/H.264 over all QP settings.

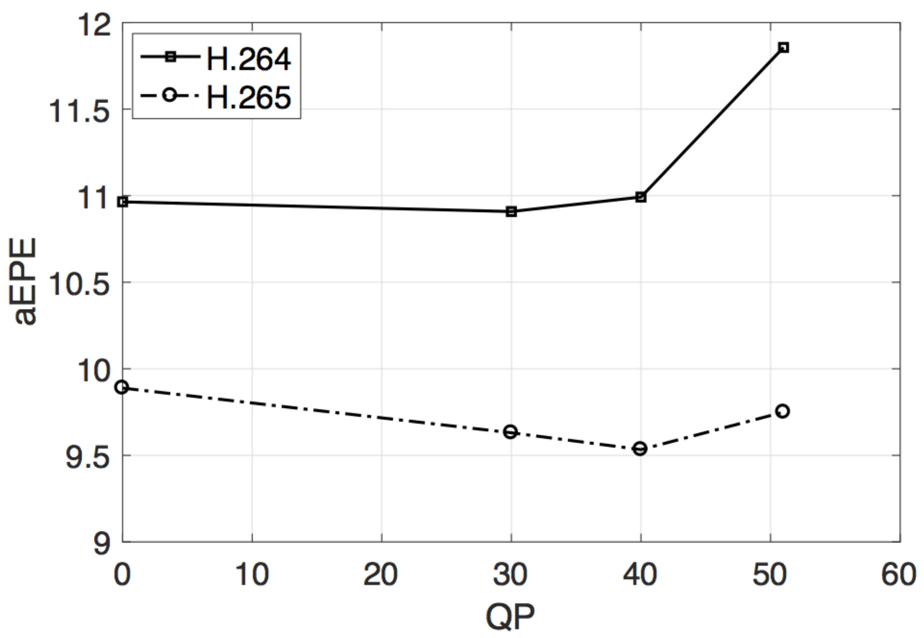

Figure 4.9: Average EPE between our approximated optical flow with different QP settings and an estimated dense optical flow ground truth using the method of Brox *et al.* [208].

4.4.3 Comparison Against External Benchmarks

In Table 4.3, we report the accuracy of our fused spatio-temporal classifier of Fig. 4.1, wherein the predictions of the spatial and temporal classifiers are averaged, and compare against state-of-the-art methods from the literature. Our results show that our approach remains competitive to the state-of-the-art on UCF-101, while retaining the significant bitrate gains reported in Table 4.1 and Table 4.2. In addition, while our approach is outperformed by methods like ST-ResNet and TSN, it is important to emphasize that these methods are orders-of-magnitude more complex than operating with sparse compressed-domain information [195, 121, 190], since they require the use of dense optical flow and need to receive and decode entire video bitstreams. Moreover, ST-ResNet and TSN use significantly deeper neural network architectures in comparison to our approach, which makes their inference significantly more compute intensive than the CNN architectures of Fig. 4.1. Finally, in order to improve our results for the HMDB dataset, our rate-optimization method can be applied in conjunction with the recent motion vector accumulation method proposed in CoViAR [192], which uses compressed-domain information to infer a sparse optical flow representation. While their optical flow approximation method is more complex in comparison to ours, by applying our classifier selection framework to such representations it is possible to gain even more savings in bitrate.

Framework	R_{cropped}	Accuracy (%)	
	(kbps)	UCF	HMDB
3D-CNN-F (H.264, QP = 30)	112.3	88.1	53.0
3D-CNN-F (H.264, QP = 40)	49.9	88.1	52.9
3D-CNN-F (H.264, QP = 51)	20.0	84.0	47.7
3D-CNN-F (H.265, QP = 30)	58.8	86.7	50.9
3D-CNN-F (H.265, QP = 40)	26.7	86.6	50.7
3D-CNN-F (H.265, QP = 51)	9.8	81.4	47.1
EMV + RGB-CNN [190]	—	86.4	—
MVCNN [121]	—	89.8	56.0
CoViAR [192]	—	90.4	59.1
ST-ResNet + iDT [185]	—	94.6	70.3
ActionVLAD + iDT [185]	—	93.6	69.8
TSN (3 modalities)[217]	—	94.2	69.4
I3D[218]	—	93.4	66.4
TSCNN (SVM fusion) [17]	—	88.0	59.4
LTC[18]	—	91.7	64.8
C3D (3 nets)+IDT[187]	—	90.4	—

Table 4.3: Comparison of our 3D-CNN-F classifier (fusion of VGG-16 spatial CNN and 3D-CNN as shown in Fig. 4.1) against state-of-the-art CNNs. R_{cropped} is estimated from UCF-101 and is not relevant for other methods as they require full decompression of whole bitstreams.

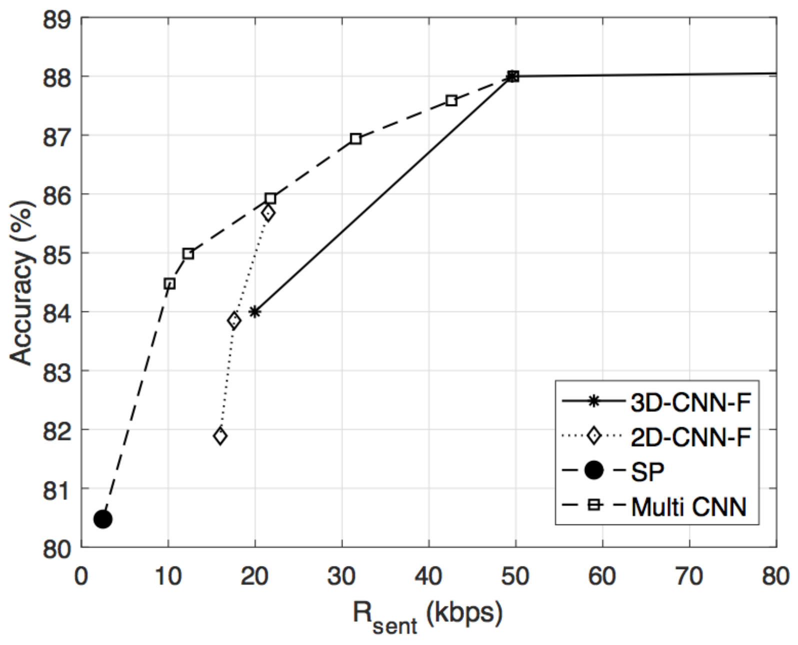

Figure 4.10: Rate-accuracy results on the UCF-101 dataset. For the 3D-CNN-F and 2D-CNN-F classifiers (fusion of spatial CNN with 3D/2D motion CNNs as shown in Fig. 4.1), different rates are obtained by using different QP settings. When using Multi-CNN, rate is controlled by setting QP = 40 and varying $R_{\text{available}}$ to solve for R^*_{L} and R^*_{H}. Note that the leftmost point shows the performance when the temporal stream is not used and the MCNN selector only considers the outputs of the spatial stream model.

4.4.4 MCNN Performance

To study the performance of our proposed MCNN under varying rate constraints, we solve (4.1) for multiple values of $R_{\text{available}}$ within the interval $[0, 50]$ kbps as described in Section 4.2.4. We then assess the MCNN accuracy on the UCF-101 test set for each set of parameters $\{R^*_{\text{L}}, R^*_{\text{H}}\}$ and show the results in Fig. 4.10. When using the optimization framework of Section 4.2.4, approximately 25 kbps (50%) reduction in bitrate can be obtained against the 3D-CNN-F classifier (25 kbps vs. 50 kbps) at less than 2% reduction in classification accuracy. Importantly, further bitrate reductions are made possible with graceful (and monotonic) degradation in classification accuracy, to the point of making it viable to get an accuracy within

7% from the top performance at an average bitrate as low as $R_{\text{sent}} = 3$ kbps. This shows the potential for further exploration of rate-accuracy optimization in CNN-based video classification and the utility of features such as R_{motion} in inferring the temporal information needed for classification.

4.5 Concluding Remarks

In this chapter we presented the first exploration of rate-accuracy trade-offs within the context of action classification via deep neural architectures. Given that our proposed method can be applied based on standardized codecs with minimal bitstream modifications, it is well suited for remote inference (e.g., for distributed internet-of-things systems), or low-complexity implementations (e.g., to run data-efficient models on mobile devices). We have observed that non-monotonic rate-accuracy curves are obtained by state-of-the-art CNNs classifying approximated flow from compressed bitstreams (following the AVC/H.264 and HEVC standards). On the other hand, a rate-based selection method between multiple CNN classifiers with varied input requirements is shown to achieve monotonic rate-accuracy characteristics. Our results show that, when reducing bitstreams to the necessary elements for 2D or 3D CNN classification, 28%-92% and 11%-94% reduction in bitrate can be achieved for AVC/H.264 and HEVC respectively. The latter observations on data redundancy detection for video classification motivated us to investigate task-agnostic data utility optimization, and in the next chapter we detail our work to that effect.

Chapter 5

Biased Mixtures Of Experts For Visual Inference Under Data Transfer Limitations

To bridge the gap between the input requirements of inference models and the practical constraints on available data per input, it is important to design models that can perform well when available communication resources are limited between the visual sensing and neural network processing parts of the system. For instance, cloud-based video analytics, remote medical imaging and robotic, drone or Internet-of-Things oriented computer vision [58, 219, 220] have stringent constraints on the amount of data that can be provided between data-producing clients and data-consuming models on cloud servers. For typical deep learning models where a fixed amount of data is required per inference task, this leads to unnecessary and often unachievable demands in the amount of required data traffic.

This chapter details our study on task-agnostic data utility optimization, in which we propose a novel class of mixtures of experts to adapt computer vision models to data transfer limitations at test time. Although some work has been devoted to input dimensionality reduction [221, 222, 166] and rate-constrained model optimization for specific tasks [58, 223], to the best of our knowledge, no task-agnostic method has been proposed that explicitly addresses data scarcity at test time by considering the variance between different domains in input space. The ex-

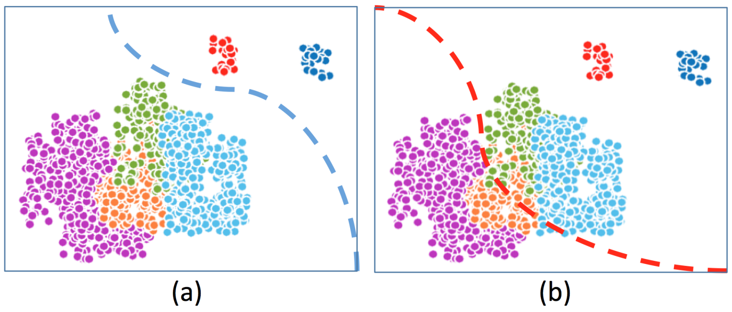

Figure 5.1: Sample space of a classification task using two features, where colours indicate different actions. All samples were drawn from a mixture model comprising gaussians with distinct coefficients of variance. The blue line of (a) shows an appropriate instance of input segregation, such that some samples can be directly classified via the use of one dimension (along the x-axis). The red line of (b) shows a randomly set partition, which does not provide any useful priors for data-economic inference.

ample of Figure 5.1 illustrates a classification task where the acceptable data cost of inference can vary for different input space partitions. That is, two features (speed and repetition of motion) can be used to classify the bottom-left examples in Figure 5.1, while one feature suffices for distinguishing "Jog" examples from "Run" examples on the top-right. Reducing the retained dimensions directly correlates with the *data cost* of inference. To leverage inherent variances across different input space partitions, and by selecting among two experts E_1 and E_2 which respectively require d_1 and d_2 bytes per input where $d_1 > d_2$, decision boundaries can be determined to appropriately pass more data for more difficult inputs. Learning decision boundaries similar to those of Figure 5.1 can allow sensors to remotely communicate data as necessary, subject to the general position of an input within its respective space. This reduces the overall data cost for inference that is accurate enough for the task at hand. Consequentially, this can relieve unnecessary load on communication resources that exist between sensors and remote machines used for visual inference. Our work proposes a solution to learning such decision boundaries directly from data for any model wherein inputs can be sub-sampled or reduced, and for any specified limit on data cost. We summarize our contributions to the following:

1. We introduce a novel class of mixtures-of-experts, wherein some experts are favored to others by design. When experts of different data requirements are included, this allows mixtures to meet different constraints on allowed data utility.

2. We propose two methods to train biased mixtures such that input space is effectively partitioned for each expert to realize data-efficient mixtures.

3. We show that data transfer optimization between visual sensing and processing can be formulated as a convex optimization problem, and present an ablation study of the benefit of biased mixtures under different contexts of allowed limits on data utility.

In using different of instances of proposed biased mixtures we task-agnostically consider the *data cost* optimization problem, and do so in order to determine required input volumes and code lengths prior to visual inference. We consider how input space partitions vary in the amount of data required per input in order to ensure good performance, and leverage this variance to train more data efficient mixtures of experts. To do so, we take inspiration from recent work [40, 39, 57] to propose a sparse mixture of experts where expert utility is biased towards specific experts. While meeting predefined constraints on expert utility bias, we train sparse gating functions to select the most adequate expert to use from a set of experts of varied input requirements. Importantly, our method does not modify any pre-existing methods for complexity optimization or task specific data cost reduction. As such, our proposal can be applied in conjunction with recent proposals on learnable compression [44, 169, 170] and domain adaptation [173, 174, 175] to reduce the data cost of visual inference.

The expert utility biasing method proposed in this chapter can be applied to reduce the data cost of any model wherein the size of inputs can be sub-sampled or reduced. To show this, we train and validate on a variety of tasks spanning multiple domains. Specifically, we validate on the tasks of: single shot object detection from the work of Wei *et. al* [22], realtime video action classification from the work of

Zhang *et. al* in [59] and Chadha *et. al* [121], and image super resolution from the work of Shi *et. al* [46] and Dong *et. al* [224]. Sections are organized as follows: Section 5.1 details the proposed biased expert selection and describes its general architecture and how it is trained. In Section 5.2 we evaluate the performance of the proposed method on all tasks, and illustrate the benefits that biased mixtures of experts can provide on multiple models for each task. Finally, Section 5.4 summarises our findings and concludes this chapter.

5.1 Biased Expert Selection

5.1.1 General Architecture Formulation

Let $\mathcal{E}$ denote a mixture of N experts where $\mathcal{E} = \{E_1, E_2, ..., E_N\}$, and each expert E_n is a modified variant of a task-performing baseline model. Per input $\boldsymbol{x}$, a gating function determines the contribution of the n^{th} expert as:

$$G(\boldsymbol{x}; \mathcal{W}_g)_n = \frac{e^{f(\boldsymbol{x}; \mathcal{W}_g)_n}}{\sum_{m \neq n} e^{f(\boldsymbol{x}; \mathcal{W}_g)_m}} \tag{5.1}$$

where $\mathcal{W}_g$ is a set of trainable weight parameters, m denotes remaining gate indices, and $f(\boldsymbol{I}; \mathcal{W}_g) \in \mathbb{R}^N$ is the output of a specified gating model (e.g, a multi-layer perceptron). The output $\boldsymbol{y}$ of the mixture of experts is:

$$\boldsymbol{y} = \sum_{n=1}^{N} G(\boldsymbol{x}; \mathcal{W}_g)_n E_n(P_n(\boldsymbol{x})) \tag{5.2}$$

where P_n is a preprocessing function to accommodate $\boldsymbol{x}$ for the n^{th} expert (e.g., P_n performs subsampling if E_n ingests subsampled inputs). Mixtures of experts are typically trained using a task loss that calculates the error between a provisioned ground-truth and $\boldsymbol{y}$. In our proposed *biased* mixtures of experts, experts are activated only when needed, and activating some experts is more favorable to activating others. In addition, all experts are optimized before training the mixture, and the training loss is back-propagated through the gating function exclusively during training. In Figure 5.2 we illustrate some examples of how biased mixtures can be applied for different tasks. To adjust mixtures for biased expert selection, we denote

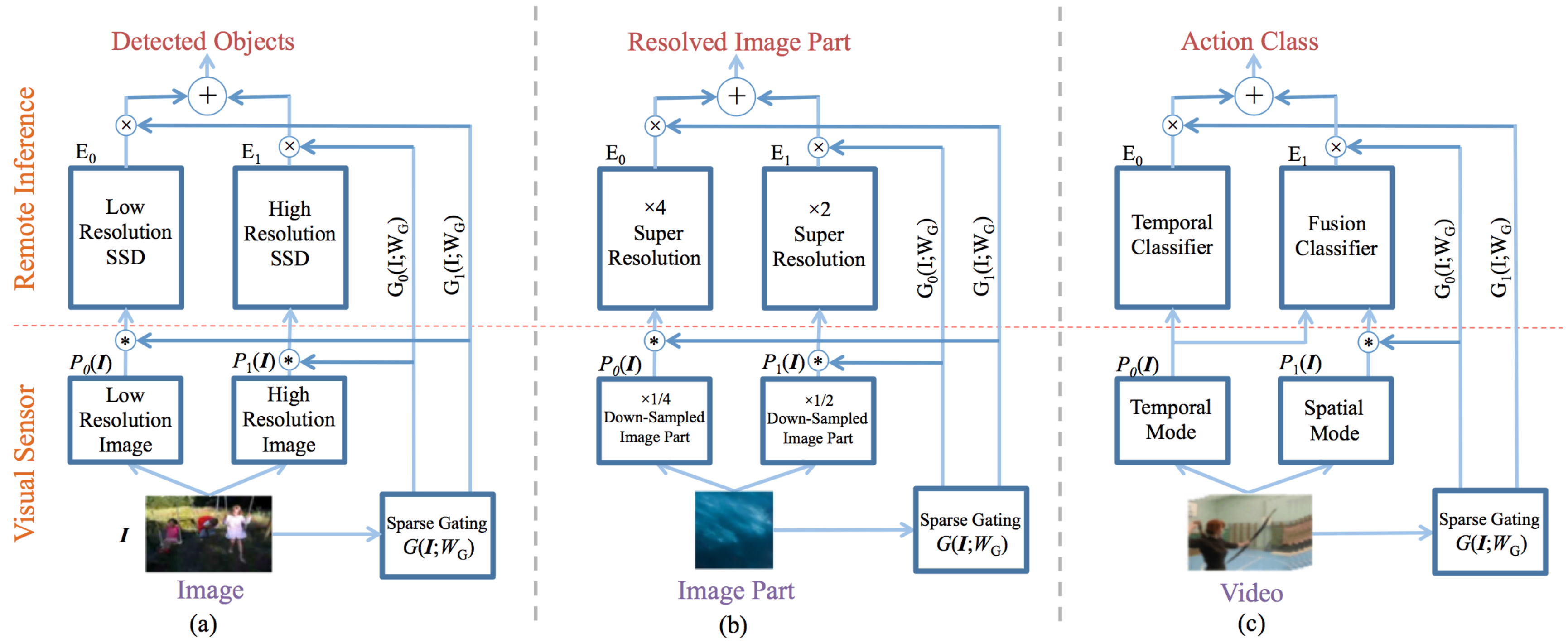

Figure 5.2: An illustration of how biased mixtures of experts can applied for different computer vision tasks. ($*$) is a special operator that transmits data to remote inference parts of the model whenever it receives a non-zero gate value. From left to right: (a) single shot detection (SSD), (b) image super resolution, and (c) realtime action classification

the desired amount of bias in expert selection by $\boldsymbol{b} \in \mathbb{R}^N$, where $||\boldsymbol{b}||_1 = 1$ and each of its components b_n specifies per batch the ratio of input examples to pass to each n^{th} expert. We then consider two methods of training for biased expert selection, and detail their function in the following.

5.1.2 Soft Bias Regularization

Here we consider a soft regularization approach, where the most suitable expert to use is selected *per input* via a sparse gating function, and all other experts are omitted. To do so, similar to Shazeer *et. al* [40], for each input $\boldsymbol{x}$ only experts associated with the highest gate value are retained for inference, and we modify the gating function to:

$$G(\boldsymbol{x};\mathcal{W}_g)_n = \psi(f(\boldsymbol{x};\mathcal{W}_g))_n \cdot \frac{e^{f(\boldsymbol{x};\mathcal{W}_g)_n}}{\sum_{m\neq n} e^{f(\boldsymbol{x};\mathcal{W}_g)_m}} \tag{5.3}$$

where $\psi(\boldsymbol{f}(\boldsymbol{I};\mathcal{W}_g))$ is a non-linear operator which returns a one-hot vector indicating the top value in $f(\boldsymbol{I};\mathcal{W}_g)$. From (5.3) we also define the utility of each n^{th} expert $\boldsymbol{u}_n$ as its total contribution per batch $\mathcal{X}$ comprising M examples:

$$\boldsymbol{u}_n = \frac{1}{M}\sum_{\boldsymbol{x}\in\mathcal{X}} G(\boldsymbol{x};\mathcal{W}_g)_n \tag{5.4}$$

and we calculate the bias regularization loss l_{bias} as a function of $\boldsymbol{u} \in \mathbb{R}^N$ and the specified bias vector $\boldsymbol{b}$:

$$l_{\text{bias}} = -w_{\text{bias}} \log(1 - \frac{1}{\sqrt{2}}\,||\boldsymbol{u} - \boldsymbol{b}||_2\,) \tag{5.5}$$

where w_{bias} is a hyperparameter to control the amount of bias to impose on the mixture. The distance is normalized by $\sqrt{2}$ to ensure the expression within the log function is always positive ($\sqrt{2}$ is the maximum possible distance between vectors with an L_1 norm of one, which is the case for $\boldsymbol{u}$ and $\boldsymbol{b}$). By applying the modifications to the gating function in (5.3), and including the bias regularization loss in (5.5) to the total loss, the mixture of experts is simultaneously trained to maximize task performance and meet the specified bias.

5.1.3 Batchwise Bias Enforcement

In our second proposal, rather than encourage mixtures to align the utility of their experts with the specified bias, we enforce bias *per batch* in accordance with $\boldsymbol{b}$, and train the mixture only with respect to its task loss. This in effect trains mixtures to make better expert selections for each input, while meeting the bias constraint for every batch. Specifically, with a batch size of M, batches are segmented such that Mb_n examples are passed to each n^{th} expert. To do so, starting from (5.1), we consider $G \in \mathbb{R}^{M \times N}$ as an M sized batch of gate vectors $G(\boldsymbol{x};\mathcal{W}_g)$, and perform the procedure described in Algorithm 1. For each n^{th} expert, we denote gate values assigned to columns of input as $G_{:,n}$ and illustrate this in Figure 5.3.

Algorithm 1 Batchwise Bias Enforcement

```
Input:  Soft gates batch $G \in \mathbb{R}^{M \times N}$
 1: for $n = 1$ to $n = N$ do
 2:     $k \leftarrow M\boldsymbol{b}_n$
        Calculate number of inputs to pass to $n^{th}$ expert
 3:     $t \leftarrow TopK(G_{:,n}, k)$
        Find top $k$ values corresponding to $n^{th}$ expert
 4:     for $i = 1$ to $i = M$ do
 5:         if $t_i \neq 0$ then
 6:             $G_{i,j} \leftarrow 0 \;\; \forall j \neq n$
                For $i^{th}$ input, set all gates not of $n^{th}$ expert to 0
 7:         else
 8:             $G_{i,n} \leftarrow 0$
                Set gate of $i^{th}$ input and $n^{th}$ expert to 0
 9:         end if
10:     end for
11: end for
```

5.1.4 Selecting Bias For Data Cost Optimization

Here we detail our method for selecting useful biases that can optimize performance under different constraints on data utility. We consider the inference data cost vector $\boldsymbol{d} \in \mathbb{R}^N$, where each of its components d_n is the size of input volumes per example as seen by each expert (i.e., the data cost associated with $P_n(\boldsymbol{x})$). When mixtures are biased and an ample number of samples is considered, the average data cost is then

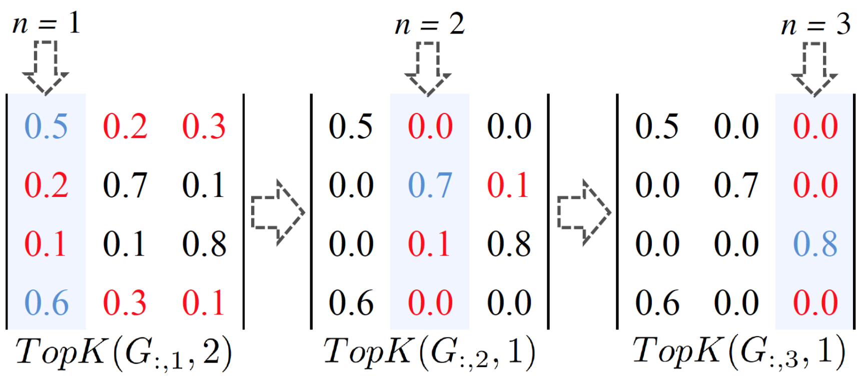

Figure 5.3: Batchwise bias enforcement example when $N = 3$, $M = 4$ and $\boldsymbol{b} = [0.50, 0.25, 0.25]$. Inputs are selected per batch by iteratively sorting and selecting the top Mb_n highest gate values. Gates subsequently set to zero are highlighted in red, and top (Mb_n) values are highlighted in blue.

expressed as $\bar{d} = \boldsymbol{bd}^{\mathrm{T}} = \sum_{n=1}^{N} b_n d_n$. In this way, the biasing vector $\boldsymbol{b}$ can be tuned to allow for different average data costs of inference in the interval $[d_{min}, d_{max}]$, where d_{min} and d_{max} are the minimum and maximum amounts of data that can be ingested by experts in the mixture.

Importantly, it can be seen that when $N > 2$ there can be multiple instantiations of $\boldsymbol{b}$ that produce the same average data cost $\bar{d}$. Thus, when an average data cost target $d_t \in [d_{min}, d_{max}]$ is specified, it is necessary to define a method by which to determine an appropriate bias vector $\boldsymbol{b}$ that is subsequently used in training biased mixtures. To address this, we consider $\boldsymbol{p} \in \mathbb{R}^N$, which quantifies the performance of each optimized expert prior to inclusion in the mixture, and select $\boldsymbol{b}$ such that: *(i)* $\boldsymbol{b}$ satisfies $\bar{d} = d_t$, and *(ii)* $\boldsymbol{b}$ maximises the expected test performance as measured by $\boldsymbol{bp}^{\mathrm{T}}$. That is, when each component p_n denotes an appropriate performance measure for the n^{th} expert on a designated set of inputs isolated from testing examples (e.g., p_n can be accuracy for classification tasks, or mean average precision for objection detection tasks), $\boldsymbol{bp}^{\mathrm{T}}$ is a probabilistic measure of performance when examples are randomly assigned to experts with respect to $\boldsymbol{b}$. In doing so, we reduce the problem of determining $\boldsymbol{b}$ for a specified data cost d_t to a linear optimization problem that achieves $\boldsymbol{bd}^{\mathrm{T}} = d_t$, while maximising $\boldsymbol{bp}^{\mathrm{T}}$. Since $||\boldsymbol{b}||_1 = 1$ and b_N can be expressed as $b_N = 1 - \sum_{n=1}^{N-1} b_n$, by expanding and substituting b_N, we get:

$$b_1 d_1 + b_2 d_2 + ... + (1 - \sum_{n=1}^{N-1} b_n) d_N = d_t \tag{5.6}$$

and following that components of $\boldsymbol{b}$ must be summable to unity, we also get the additional $(N-1)$ constraints:

$$b_1 \leq 1;\ b_2 \leq 1; ...\ ;\ b_{N-1} \leq 1 \tag{5.7}$$

with maximization objective:

$$\max\{b_1 p_1 + b_2 p_2 + ... + b_N p_N\} \tag{5.8}$$

Note that (5.6) and (5.7) define N linear constraints to maximize the objective (5.8) with N basic values $\{b_1, b_2, ..., b_N\}$, and determining $\boldsymbol{b}$ is a convex problem which can be readily solved by optimization methods such as the simplex method [225]. Thus, an appropriate biasing value $\boldsymbol{b}$ to use for training can be found for any specified target data cost d_t. Following the duality property of such convex problems, we can also formulate the equivalent problem that finds $\boldsymbol{b}$ for any specified performance target p_t.

5.1.5 Additional Observations On Biased Mixtures

Note that while (5.5) and Algorithm 1 do not directly consider data cost, by using a set of experts that require different amounts of data, the bias vector $\boldsymbol{b}$ controls the average data cost per batch. For example, the mixture of experts can be encouraged to pass data more economically by setting $\boldsymbol{b}$ to favor the utility of some data efficient experts over others. Importantly, the quality of expert selections is related to the complexity of the gating function $G(\boldsymbol{x}; \mathcal{W}_g)$; increasing the complexity of $G(\boldsymbol{x}; \mathcal{W}_g)$ can improve selections, albeit with dimnishing returns. In addition, because bias enforcement is done per batch, we intuitively expect the quality of selection to be directly correlated to the batch size: setting a small batch size may not expose the gating function to a sufficient amount of variance in inputs to make selections of benefit, while increasing the batch size at test time is favorable.

The same expert selection methods described here can be applied with mixture architectures that include experts which are optimized for low data cost via dimensionality reduction methods (e.g., the proposals of [58, 59]) and experts that use different modalities to make their inferences (as illustrated in (c) of Figure 5.2). Moreover, while our work studies the problem of reducing data cost, $\boldsymbol{b}$ can also be specified to prioritize any other expert property to meet any constraint (e.g., to meet constraints on power consumption or latency).

5.2 Evaluation

5.2.1 Benchmarks And Evaluation Method

To show how biased mixtures can optimize data costs of inference for different problems, we evaluate on three computer vision tasks: *(i)* object detection, *(ii)* image super resolution, and *(iii)* realtime action classification. In reporting results for all tasks, we compare our method against two alternatives:

1. *Previously Proposed Models*: To benchmark our results against relevant task-specific solutions, we consider the performance of constituent experts when optimized for different data cost constraints. In biased mixtures, this corresponds to specifying $\boldsymbol{b}$ as a one hot vector, and measures performance when the same amount of data is used for all inputs during inference (e.g., when $\boldsymbol{b} = [0, 1, 0]$ only E_2 is used for inference). We report this to benchmark against previous work and to highlight the benefit of uniquely dividing the input space for each expert.

2. *Random Selection*: Here, experts are randomly selected for inference at test time in order to satisfy the model biasing requirement $\boldsymbol{b}$. This is to serve as the lower bound of performance when biased mixtures are used and the specified expert utility bias is met.

Importantly, when considering the problem of task-agnostic model optimization under data cost constraints, there is no previous work similar to ours. That is why, we benchmark against the maximum performance achievable by recently proposed *task-specific* solutions when their input volumes are adjusted to meet different constraints on data cost. That is, *biased mixtures consist of experts that also stand in as external benchmarks*. To highlight the latter, benchmark results of constituent experts are indicated in comparative plots by markers on dotted lines.

For clarity, and to ensure consistency of representation across all tasks, we report the per input data cost of inference $\bar{d}$ as the average amount of data seen by the mixture after inputs are fully decompressed.For each evaluated task we specify how the data cost for each expert d_n is measured (i.e., the data cost associated with $P_n(\boldsymbol{x})$). For a concise measure of how well models preform across different specified data cost constraints of $d_t \in [d_{min}, d_{max}]$, and with $p_{test}(d_t)$ denoting test performance when the target data cost is d_t, we report the area under curve when data cost is normalized as:

$$\rho = \int_0^1 p_{test}(d_{min} + t(d_{max} - d_{min}))\, dt \tag{5.9}$$

For all mixtures, we specify the gating model (i.e., $f(\boldsymbol{x}; \mathcal{W}_g)$) as a single conv-pool layer followed by a fully connected network. To ensure that the model selection process is of low complexity for all tasks, we use ReLU activated depth-wise separable convolutions [226], and report the per input number of multiply-accumulate gating operations C_g. We use cross-validation to optimize the biasing weight w_{bias} and report the best performance when soft regularization is used. After all experts included in the mixture are individually optimized, biased mixtures are trained by updating the weights of the gating function exclusively, and the weights of experts are not fine-tuned further. We have found that using higher batch sizes is helpful when training biased mixtures, because it exposes the mixture to a more varied set of input examples to partition to each expert meaningfully. Therefore, to ensure gating functions learn meaningful features for batch partitioning, for all tasks we set the batch size to 128 and the learning rate to 10^{-4}.

5.2.2 Single Shot Object Detection

We test our method on single-shot detection (SSD) to reduce the data requirement for object detection while maintaining high accuracy. Recent work [227, 57] showed that SSD models [228, 229, 230, 231, 232] vary widely in performance and complexity when input sizes are adjusted. When considering the varying degrees of complexity of natural images, we expect that the minimum required subsampling rate of inputs for accurate object detection should vary accordingly. To demonstrate this, we train a biased mixture of experts where each expert is optimized for a different image subsampling rate, and use the recent work of Liu *et. al* [22] as a baseline for all experts (for an illustration, see (a) of Figure 5.2). When the resolution of inputs to each expert is $R_n \times R_n$ pixels, we measure the data cost associated with $P_n(\boldsymbol{I})$ as $3 \times R_n \times R_n \times K$, where 3 is the number of color channels in RGB inputs, and K is the number of bytes needed to store floating point decimals.

We use VGG16 [185] and ResNet50 [233] for feature extraction and evaluate all models using 300 regional proposal boxes per image. Following recent work [227, 22], we train on COCO training data while excluding the 8k mini-eval images used in the 2012 challenge [234], and report performance as the mean Average Precision (mAP) on COCO (07+12). We train mixtures for 20k steps to show our results when using soft regularization and bias enforcement, and ensure that the gating complexity of all mixtures remains at $C_G < 10^8$ Mult-Add operations.

Figure 5.4 shows the relationship between imposed bias, data cost, and mAP when three VGG16 experts are used for single shot detection, where the resolution of inputs to each expert is $\{R_n\} = \{100, 150, 300\}$. Notably, biased mixtures optimized with bias enforcement provide the slowest degradation in mAP for lower data costs, with diminishing gains when more data is available at test time. Specifically, biasing via enforcement outperforms individual experts by 7.5% when an average of 220 kilobytes per image is allowed, which is equal to the performance of individual experts at 490 kilobytes. That is, when the minimum acceptable mAP is 70%, a reduction of 270 kilobytes in required data is achieved by our proposal (which is equivalent to a saving of 55%).

In Table 5.1 we show the performance of biased mixtures when applied to multiple models, and report ρ as a comprehensive measure of model performance across data costs. When compared to random selection, we note that for both ResNet50

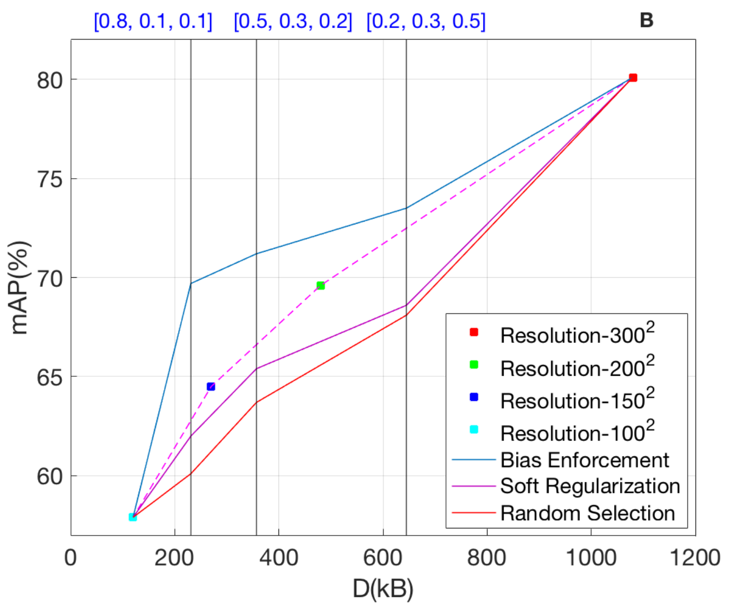

Figure 5.4: Single shot detection performance comparison of biased mixtures of VGG16 [188] experts against other benchmarks when $\{R_n\} = \{100, 150, 300\}$. The performance of individual experts is shown on the dotted line.

[233] and VGG16 [188], imposing bias on mixtures provides the highest gain when lower values of data cost are considered (e.g., when $\bar{d} < \frac{d_{max}}{4}$).

Compared to soft regularization, and for all mixture configurations, we found that bias enforcement is a much more effective method for training biased mixtures (this is also true for all other tasks evaluated). We hypothesise this is because, when bias enforcement is used only the task loss is back-propagated during training, which causes less competition between losses and therefore less local minima to exist in solution space.

Table 5.1: Single shot detection comparison on COCO [234] of biased mixtures of SSD [22] experts against other benchmarks. Resolutions $\{R_n\}$ and data costs $\{D_n\}$ are reported for all experts.

$\{R_n\}=\{100, 150, 300\}$(Pixels); $\{d_n\} = \{120, 270, 1080\}(kB)$

Feature Extractor	Biasing Method	mAP($\bar{d}$) (%) when $\bar{d} =$ d_{max}	$\frac{d_{max}}{2}$	$\frac{d_{max}}{3}$	ρ
VGG16 [188]	Optimized Experts	80.0	70.0	66.7	70.9
	Bias Enforcement		**72.5**	**70.9**	**73.1**
	Soft Regularization		67.1	65.0	68.9
	Random Selection		66.3	63.4	68.2
ResNet50 [233]	Optimized Experts	75.7	65.1	61.3	66.1
	Bias Enforcement		**67.8**	**65.9**	**68.3**
	Soft Regularization		62.2	59.9	64.2
	Random Selection		61.9	57.4	63.3

In Table 5.2 we study the effect of adjusting the gating complexity C_G, batch size M, and number of experts N on the performance of biased mixtures when bias enforcement is used. When we consider all mixtures, we find that batch size is critical to performance. This is because bias is enforced on a per batch basis, and to make meaningful decisions the gating function needs to be exposed to an ample amount of variance variance between examples. We also see that increasing the complexity of gating does increase performance by helping partition the input space more effectively. However, this effect saturates at $C_G \approx 3.8 \times 10^7$ Mult-Add operations, which demonstrates that the optimal hyperplane to partition input space for $N \leq 3$ experts can be learned with low complexity.

Table 5.2: Relation between gating complexity, batch size, and performance when bias enforcement is used.

C_G (Mult-Adds)	M	ρ when $\{R_n\} =$ $\{100, 300\}$(Pixels) VGG16 [188]	ResNet[233]	$\{100, 150, 300\}$(Pixels) VGG16 [188]	ResNet50[233]
23,048,576	16	68.40	64.11	69.27	64.46
	32	70.35	65.89	70.25	65.57
	64	**70.93**	**66.24**	**71.16**	**65.72**
26,194,304	16	70.85	66.92	71.82	67.04
	32	71.49	67.25	72.50	67.41
	64	**71.84**	**67.59**	**72.97**	**68.04**
38,700,216	16	70.93	67.01	72.10	67.33
	32	71.58	67.25	73.07	68.26
	64	**71.86**	**67.62**	**73.13**	**68.30**

By comparing the left and right part of Table 5.2, we see that adding more experts to the mixture provides a modest increase to performance. This is because having more experts allows the mixture to further exploit the variance in different input sub-spaces (if any such variance exists). To see the extent to which this is true, in Figure 5.5 we adjust the limits of allowed input resolutions to the mixture R_{min} and R_{max}, and report ρ when considering different values of N. Importantly, we see that when the difference between R_{min} and R_{max} is lower, using more experts yields less gain in performance, to the point where using more than three experts for $(R_{min}, R_{max}) = (100, 300)$ does not provide any benefit. This is because, while setting high values of N increases the number of intermediate resolutions between R_{min} and R_{max}, if $R_{max} - R_{min}$ is low the amount of discernable adequacy between experts is also low, which in turn diminishes the benefit of including more experts.

5.2.3 Image Super-Resolution

We test the applicability of biased mixtures on Single Image Super resolution (SISR) [30, 236, 237, 238], an image reconstruction tasks where spatial features of high-resolution images are inferred from low-resolution input images. Several recent proposals have shown good performance in terms of image reconstruction accuracy and computational efficiency [46, 224, 239].

Table 5.3: Image super resolution comparison on DIV2K [235] of biased mixtures and other benchmarks. Upscale factors $\{S_n\}$ and data costs $\{D_n\}$ are reported for all experts.

$\{S_n\}=\{\times2,\times3,\times4\};\ \{d_n\}=\{13.9,21.8,49.2\}(kB)$

Model	Biasing Method	PSNR($\bar{d}$) (dB) when $\bar{d}=$			ρ
		d_{max}	$\frac{d_{max}}{2}$	$\frac{d_{max}}{3}$	
ESPCN[46]	Optimized Experts	33.3	30.4	28.4	30.7
	Bias Enforcement		**30.7**	**28.8**	**31.0**
	Soft Regularization		30.0	28.1	30.6
	Random Selection		29.8	28.0	30.5
F-SRCNN [224]	Optimized Experts	32.8	29.8	28.0	30.3
	Bias Enforcement		**30.1**	**28.3**	**30.5**
	Soft Regularization		29.3	27.6	30.1
	Random Selection		29.2	27.5	30.0

However, current super resolution models do not take into account the variable amount of high-frequency edge content between images. That is, when reconstructing images which contain many high frequency elements, SISR models are likely to benefit from higher resolution input images, while images comprising predominately low-frequency content can be inferred just as well from lower resolution inputs. This is true also when considering different parts of an image, which usually vary in the breadth of their frequency elements.

To demonstrate this, we evaluate on the NTIRE17 challenge dataset DIV2K [235], and train biased mixtures to determine the needed input resolution for good image reconstruction. To expose biased mixtures to the intra-image variance of frequency elements, images are divided using a fixed grid into parts of size 64×64 pixels, and super-resolution is performed on each part separately (for an illustration, see (b) of Figure 5.2). By inspecting the low-level semantics of each image part, the mixture selects the most data efficient expert for reconstruction to preform an upscaling from the set $\{S_n\}=\{\times2,\times3,\times4\}$. For each expert that upscales inputs with a factor of S_n to match the target resolution of 64×64 pixels, we measure the

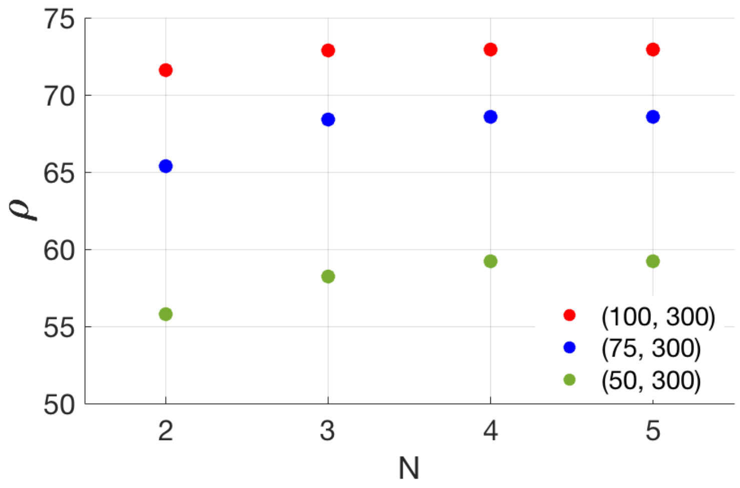

Figure 5.5: ρ when bias enforcement is used and the number of experts N is configured. VGG16 [188] is used for feature extraction, and different colors indicate the resolution limits (R_{min}, R_{max}) allowed to the mixture (where N determines the number of intermediate input resolutions included).

associated data cost as $d_n = (64/S_n)^2 \times K$, where K is the number of bytes needed to store floating point decimals. For all biased mixture results, mixtures are trained for 20 epochs and we ensure the complexity of the gating function is set to $C_G < 10^7$ Mult-Add operations.

In Table 5.3 we compare biased mixtures against other benchmarks when using ESPCN [46] and FRSCNN [224] as expert baselines, and in Figure 5.6 we show the relationship between average data cost and PSNR when considering ESPCN [46]. Notably from Figure 5.6, when bias enforcement is used and $\bar{d}$ is within the range of 18-22 kilobytes, biased mixtures outperform single experts with an average difference of 0.4 dB. Over the same range of values of $\bar{d}$, and when compared to random selection, bias enforcement provides an average improvement of 0.7 dB. This highlights the magnitude of intra-image high variance in required input resolution for image reconstruction, which is not considered by neither random selection nor optimized experts. Overall, Figure 5.6 and Table 5.3 show that biased mixtures outperform single experts most when $\bar{d} < 20$ kilobytes, with diminishing gains in performance for higher values of $\bar{d}$.

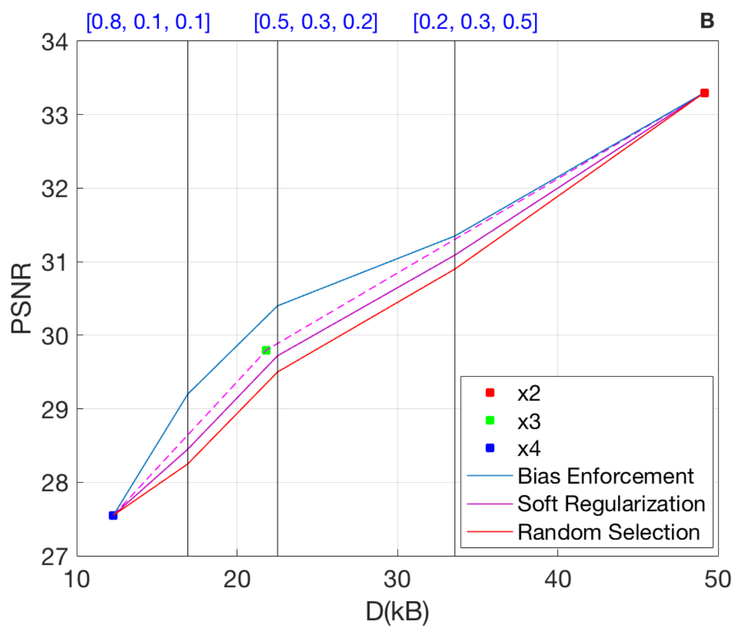

Figure 5.6: Super resolution performance comparison of biased mixture of ESPCN [46] experts and other benchmarks when $\{S_n\} = \{\times 2, \times 3, \times 4\}$.

In Figure 5.7 we show examples of expert selections made by the biased mixture to resolve different 64×64 inputs when bias enforcement is used. The mixture learns to pass image parts with high frequency components to the $\times 2$ SISR model, and passes other less demanding parts to the $\times 4$ model (which are blurrier, due to the lower frequency of their components).

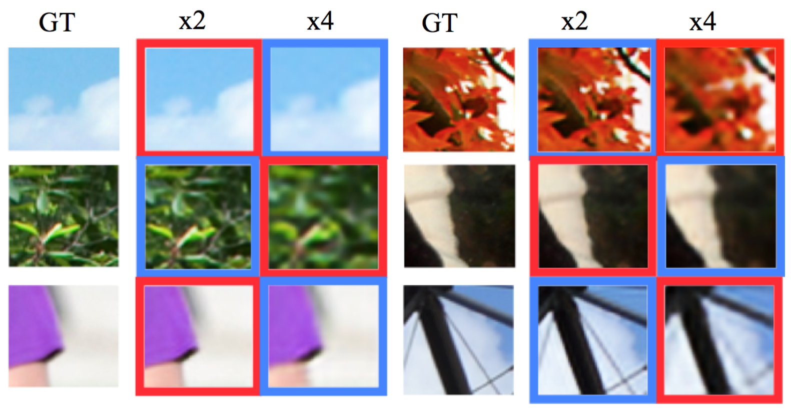

Figure 5.7: Examples of expert assignments to different image parts. Selected and non-selected experts are respectively highlighted by blue and red borders.

Table 5.4: Realtime action classification on UCF-101[184] of biased mixtures of experts and other benchmarks. Modalities $\{M_n\}$ and data costs $\{D_n\}$ are reported for all experts.

$\{M_n\}= \{\text{Temporal}, \text{Spatial}, \text{Fusion}\}$; $\{\text{d}_n\} = \{737.3, 1843.0, 2580.5\}(kB)$

Model	Biasing Method	Accuracy($\overline{d}$) (%) when $\overline{d} =$			ρ
		d_{max}	$\frac{d_{max}}{2}$	$\frac{d_{max}}{3}$	
MV-3DCNN[121]	Optimized Experts	88.0	79.0	77.9	80.9
	Bias Enforcement		**82.0**	**80.4**	**83.5**
	Soft Regularization		80.3	78.0	81.9
	Random Selection		78.8	77.3	81.3
EMV-CNN [59]	Optimized Experts	85.6	76.6	75.5	78.7
	Bias Enforcement		**80.2**	**79.2**	**81.3**
	Soft Regularization		77.2	75.6	79.7
	Random Selection		75.7	74.9	79.0

5.2.4 Realtime Action Classification

We test our method on realtime video action classification in the compressed domain. While the best performing action classification models operate on uncompressed video data, to reduce latency, the models proposed in recent work [121, 59] infer a low-resolution optical flow from codec motion vectors at high speeds for action classification. The classifiers of [121, 59] use two-stream architectures to infer actions, where spatial and temporal classifiers complement each other by learning different sets of features from their respective domains [29]. As such, for some action subsets, the use of only the temporal or spatial classifier can suffice in drawing accurate distinctions between actions, but combining the predictions of both provides the highest accuracy.

To show that biased mixtures can select among different modalities to reduce the data cost of action classification, we train a multi-modal biased mixture of experts using the models proposed in [121] and [59] as baselines (and we illustrate this in (c) of Figure 5.2). We evaluate on UCF-101 [184], and measure the cost associated with the spatial mode as $F_s \times W_s \times K \times 3$, where $F_s = 2$ is the number of RGB

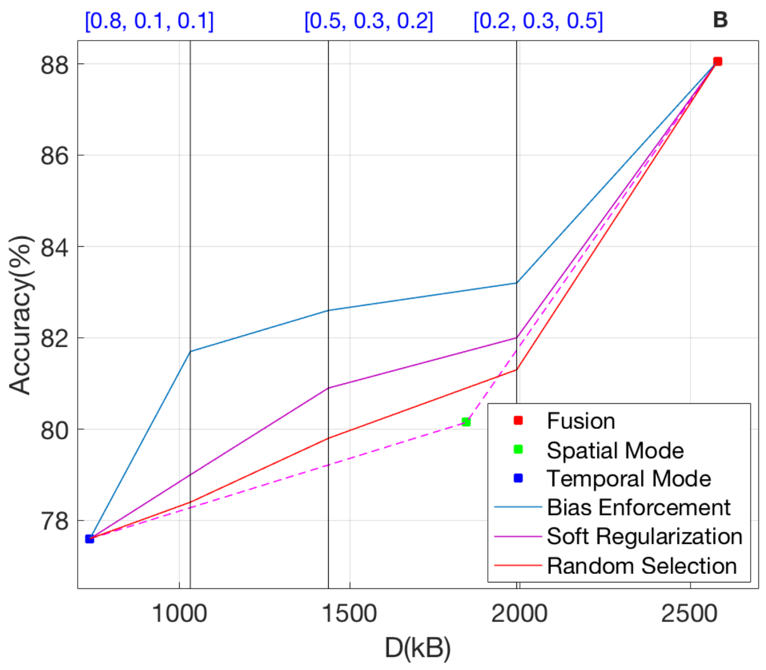

Figure 5.8: Realtime action classification performance comparison of biased mixtures of MV-3DCNN [121] experts, with expert modalities $\{\text{Mode}_n\}$=$\{$Temporal,Spatial, Fusion$\}$.

frames used, $H_s = 360$ and $W_s = 240$ are the height and width of inputs, and $K = 32$ is the number of bytes to store floating point decimals. For the temporal model, we measure the data cost as $F_t \times H_t \times W_t \times K \times 2$, where $H_t = 24$ and $W_t = 24$ are the height and width of approximated optical flow, and $F_t = 150$ is the number of frames used (two channels are used in optical flow to represent vertical and horizontal motion). The fusion classifier uses both modalities to predict actions and is the most accurate, but requires a data cost equal the sum of both modalities. We include all modalities to train a mixture of experts $\{M_n\} = \{\text{Temporal}, \text{Spatial}, \text{Fusion}\}$, and train a gating function to select the most suitable modality to use for each input. For all biased mixtures, we train for 80k steps and restrict the complexity of the gating function to $C_G < 10^8$.

In Table 5.4 we compare the performance of biased mixtures against other benchmarks when using the spatial and temporal classifiers of [121] and [59] as baselines for experts. We first note that, both biasing methods outperform random

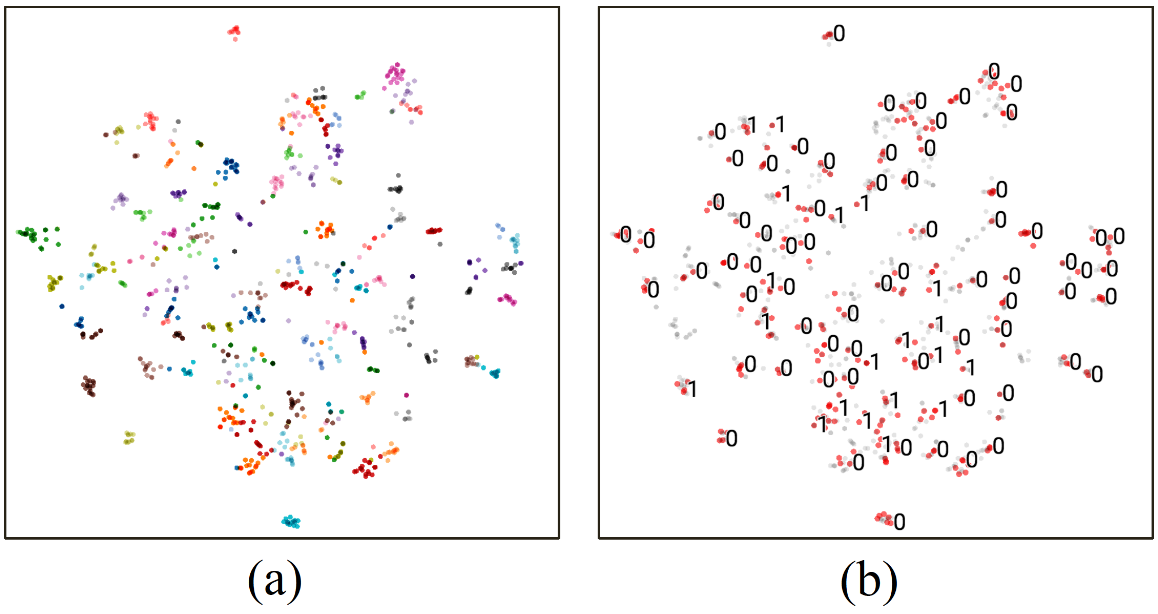

Figure 5.9: t-SNE [240] projections of 1024 UCF101 videos, where in (a) colours indicate different classes, and (b) mode assignments are shown as 0 or 1 for the temporal and fusion classifiers respectively. (zoom in to view in high-resolution)

selection, by up to 1% for soft regularization and up to 3.8% for bias enforcement. This indicates that the biased mixture learns to discern confusing classes for particular modalities to pass them to others. Notably, when $\bar{d} = \frac{d_{max}}{3} = 860$ kilobytes, bias enforcement gives an accuracy 1.4% higher than that of the optimized experts at $\frac{d_{max}}{2} = 1290$ kilobytes, which requires 430 kilobytes more in data cost.

In Figure 5.8 we show the relationship between $\bar{d}$ and action classification accuracy when a biased mixture of MV-3DCNN [121] experts is used and the mode of each expert is $\{\text{Mode}_n\} = \{\text{Temporal}, \text{Spatial}, \text{Fusion}\}$. We first note that, due to the low resolution of its inputs, the temporal classifier requires the least amount of data and can predict actions with an accuracy of 77.8%. By selecting among the three modes both biasing methods outperform random selection, by up to 3.4% for bias enforcement when $\bar{d} = 1032$ kilobytes, and up to 1.1% for soft regularization when $\bar{d} = 1438$ kilobytes. Notably, when using the temporal classifier for 80% of videos at $\bar{d} = 1032$ kilobytes (i.e., when $\boldsymbol{b} = [0.8, 0.1, 0.1]$), bias enforcement is 1.6% more accurate than the spatial classifier (which requires 811 kilobytes more in data, equivalent to an increase of 78%).

To visualize how different modalities are assigned to videos, in Figure 5.9 we show two-dimensional t-SNE [240] projections of 1024 UCF101 examples as embedded by the last layer of the temporal classifier. For clarity of presentation, we use a biased mixture of two modalities $\{M_n\} = \{\text{Temporal}, \text{Fusion}\}$ and set $\boldsymbol{b} = [0.75, 0.25]$. In this way, we show the relation between different class labels and assigned modalities. Notably, the biased mixture learns to favor using the temporal classifier for video clusters that are comparatively isolated, and are easier to discern from other clusters. For videos that are not clearly clustered or isolated (which are mostly located in the middle), the biased mixture selects the fusion model.

5.3 Details Of Used Gating Models

For all mixtures in the evaluation section, we mention that the gating model $f(\boldsymbol{I}; \mathcal{W}_g)$ is specified as a conv-pool layer followed by a fully connected network. Here in Table 5.5, 5.6, and 5.7 we detail the parameters of all gating layers for our biased mixture results on all evaluated tasks. We also note that:

1. We use ReLU activated depthwise separable convolutions [226] to reduce the complexity of gating.

2. In Table 5.5 on single shot detection, input to the gating model is center cropped to be a 224×224 RGB image.

3. In Table 5.6 on image super resolution, input to the gating model is not sub-sampled (i.e., it is the 64×64 image part before downsampling via bicubic interpolation). This is to expose gating to the high frequency components of input images.

4. In Table 5.7 on action classification, input to the gating model is the temporal mode of the video (i.e., the approximated optical flow from codec motion vectors).

Table 5.5: Layer complexity C of gating model $f(\boldsymbol{x}; \mathcal{W}_g)$ for biased mixtures evaluated on single shot detection. Expert input resolutions are specified as $\{R_n\} = \{100, 150, 300\}$ and $N = 3$.

Layer Type	Filter Shape	Stride	Input Shape	C (Mult-Adds)
Convolutional	$3 \times 3 \times 3 \times 64$	2	$224 \times 224 \times 3$	$2,747,136$
Avg. Pooling	7×7	5	$111 \times 111 \times 64$	—-
Flatten Op.	–	–	$21 \times 21 \times 64$	—-
Fully Connected	28224×1024	–	1×28224	$28,901,376$
Fully Connected	1024×3	–	1×1024	3072

Table 5.6: Layer complexity C of gating model $f(\boldsymbol{x}; \mathcal{W}_g)$ for biased mixtures evaluated on single image super-resolution. Expert upscaling factors are specified as $\{S_n\} = \{\times 4, \times 3, \times 2\}$ and $N = 3$.

Layer Type	Filter Shape	Stride	Input Shape	C (Mult-Adds)
Convolutional	$3 \times 3 \times 3 \times 64$	2	$64 \times 64 \times 3$	$224,256$
Avg. Pooling	3×3	2	$21 \times 21 \times 64$	—-
Flatten Op.	–	–	$10 \times 10 \times 64$	—-
Fully Connected	6400×512	–	1×6400	$3,276,800$
Fully Connected	512×3	–	1×512	1536

Table 5.7: Layer complexity C of gating model $f(\boldsymbol{x}; \mathcal{W}_g)$ for biased mixtures evaluated on realtime action classification. Expert modalities are specified as $\{\mathrm{Mode}_n\} = \{\mathrm{Temporal}, \mathrm{Spatial}, \mathrm{Fusion}\}$ and $N = 3$.

Layer Type	Filter Shape	Stride	Input Shape	C (Mult-Adds)
Convolutional	$3 \times 3 \times 320 \times 64$	2	$24 \times 24 \times 320$	$3,363,840$
Flatten Op.	–	–	$11 \times 11 \times 64$	—-
Fully Connected	7744×1024	–	1×7744	$7,929,856$
Fully Connected	1024×3	–	1×1024	3072

5.4 Concluding Remarks

In this chapter we extended the mixtures of experts paradigm to effectively partition input domains such that constraints on data availability at test time can be met. We proposed two methods for training biased mixtures of experts and evaluated their performance on multiple models for all investigated tasks. We demonstrated how biased mixture are applicable whenever constituent experts vary in their input dimensionality, and showed this on a wide range of computer vision tasks. Specifically: (i) on single shot detection, biased mixtures of SSD experts [22] outperform their constituent experts by 6% in mAP when $250 < d < 300$ kilobytes on COCO [234] , (ii) on image super resolution, biased mixtures of ESPCN experts [46] outperform their constituent experts by 0.5 in PSNR when $17 < d < 23$ kilobytes on DIV2K [235] , and (iii) on multi-class action classification, and when selecting between different modalities of texture and optical flow, biased mixtures provide at least 3% more in accuracy when $1000 < d < 1500$ kilobytes on UCF101 [184]. Our validation showed that, especially for lower ranges of allowed data cost, biased mixtures significantly outperform single experts optimized to meet the same constraints, and can be used to adapt computer vision models to data transfer limitations. We additionally showed how useful gating inferences that prioritise data economy can be realized with complexities that do not exceed 10^8 Mult-Add operations for all evaluation tasks, which are feasible to run even on embedded computation units mountable on lightweight sensors. Finally, we note that an important advantage of biased mixtures is the flexibility at which they can be applied, in that, biased mixtures do not modify their constituent experts, but rather augment their function with an input preprocessing stage that allows for data economy in inference.

Chapter 6

Conclusion And Future Work

6.1 Conclusion

This thesis detailed our study on rate-complexity constrained learnable inference machines, and our contributions thereto. In our first task-specific study in Chapter 3, we showed how data utility correlates with complexity of inference in video action classification, and proposed a model that facilitates low-complexity inference in the compressed domain. To do so, we demonstrated how optical flow can be sparsely approximated directly from codec motion vector data, and produced such approximations with the exclusive use of linear operations (i.e., restricted mainly to spatial and temporal bilinear interpolation). Our results in Section 3.4.3 show that neural networks can accurately classify videos while using inputs volumes that are directly extracted from compressed bitstreams and significantly smaller than those of previously proposed methods, which used larger volumes of texture information and optical flow approximations that are denser to ours. Finally, we presented a complexity study to show that our method achieves accuracies comparable to those of previous methods, with runtimes three orders of magnitude lower.

Our second contribution in Chapter 4 builds on the same vision task of video classification to study rate optimization. We considered visual analysis systems where the visual sensing and the CNN-based semantic analysis parts are not co-located on the same machine, and showed how bitrate, input noise, and performance are correlated in such systems subject to the quality and quantity of information used for inference. We focused our study on the AVC/H.264 standard to quantify the correlation between optical flow as approximated from motion vectors and state-of-the-art dense approximations derived from the pixel domain. We importantly noted the non-monotonous nature of this correlation, and produced a corresponding rate-accuracy profile on standard action classification datasets to inform the design of compressed-domain classifiers. In addition, and to understand the lower limits on volumes of required data for compressed video classification, we implemented a bitstream cropping method that only retains the necessary elements for optical flow approximation while maintaining full compliance with the AVC/H.264 coding standard. To further reduce required bitrates for video classification, in Section 4.2.4 we proposed a data redundancy inference method to selectively omit larger temporal extents when they are less likely to be necessary for accurate classification. In addition, and to quantify the benefit of inferring data redundancy prior to classification, we measured the bandwidth requirements of classifiers when fixed length temporal extents are used, and compared it against contexts where data redundancy predictions are performed prior to classification. Our validation in Section 4.4.4 demonstrated how inspecting input sparsity is sufficient to make informed decisions about required temporal extents for accurate video classification, thereby allowing for the savings in bandwidth we report. In observing the latter, we also showed how classifiers can meet constraints on input throughput, specifically in instances where sensors and inference models must communicate with each other by means that require communication bandwidth.

Our reported findings in Chapter 3 and Chapter 4 then motivated us to extend our work such that it is applicable to other tasks of vision, and in Chapter 5 we studied a task-agnostic solution for data utility optimization. Specifically, we proposed a novel class of mixtures of experts to adapt computer vision models to data transfer limitations at test time. We considered how input space partitions vary in the amount of data required per input in order to ensure good performance, and leveraged this variance to train more data efficient mixtures of experts. To discover and exploit such partitions in input space while meeting predefined constraints on expert utility bias, in Section 5.1 we detailed how biased mixtures are trained to select the most adequate expert to use from a set of experts of varied input requirements, subject to the inferred quantity of data required for inference. Importantly, our contribution Chapter 5 details how biased mixtures do not modify the definition or design of their constituent experts, but rather augments their architecture in such a way that they can be applied in conjunction with other propositions that modify models to reduce complexity and input dimensionality. In Section 5.2 we validated on multiple computer vision tasks to demonstrate how biased mixtures are applicable to to any set of pre-trained experts to optimize data utility, namely: single shot detection, image super resolution, and real time action recognition. For all tested applications, we showed how biased mixtures trained to meet different constraints on data utility outperform their constituent experts when they are optimized to meet the same constraints.

When considered in its whole, our work investigated the redundancies of recent proposals on computer vision to find more data-efficient models that reduce the throughput of ingested inputs. We finally note that, in the context of applied distributed visual systems, and to meet different constraints on complexity and bandwidth at test time, all of our observations and tests detailed in Chapters 2, 3, and 4 show the importance of conditioning data utility for visual inference to the local proximities and properties of inputs within their space. In other words, the importance of doing so is applicable to all presented vision tasks, and is likely to extend to other visual inference tasks in order to mitigate unnecessary burdens on communication resources and sensor hardware.

6.2 Future Work

The work presented in this thesis aimed to regulate the requirements of computer vision models for higher input dimensionality to meet rate and complexity constraints that exist in practical contexts of distributed systems for visual inference. This motivated us to study solutions that allow for the design of vision models that are capable of adapting to data transmission at runtime subject to: allowable quality of performance, complexity of inference, available communication channel states, and the content from which inferences are drawn. To realize solutions that come closer to that end, we see the following as pursuable future extensions of our work:

1. Our work in Chapter 5 can be extended to study hierarchical mixtures [144, 143] of experts for data utility optimization. This would be an investigation to find the extent to which good inferences can be made about the required amount of data for achieving different tasks

2. To further optimize the performance of compressed domain classifiers, recent proposals on adversarial discriminative domain adaptation [173] can be exploited to refine motion flow approximated from motion vectors and align it with dense approximations of optical flow.

3. Mixtures that comprise experts fine tuned on learned representations that reduce dimensionality such as those of [221, 43] can be learned jointly with gating functions. This in effect would fit sparse gating functions and learned representations to compliment each other.

4. Our study in Chapter 5 studies the question "What is the minimum amount of data to use for good inference ?", and proposes a heuristic answer; a question complementary to the latter is "Which are the most useful parts of data to use for inference ?". Studying models that answer both questions jointly is a natural trajectory of our research, and constitutes the basis of our next study.

Bibliography

[1] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2011.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[4] Aaron Chadha, Alhabib Abbas, and Yiannis Andreopoulos. Video classification with cnns: Using the codec as a spatio-temporal activity sensor. *arXiv preprint arXiv:1710.05112*, 2017.

[5] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[6] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Com-

mon objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[8] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[9] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–135, 2017.

[10] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[11] Aysegul Dundar, Jonghoon Jin, Vinayak Gokhale, Bharadwaj Krishnamurthy, Alfredo Canziani, Berin Martini, and Eugenio Culurciello. Accelerating deep neural networks on mobile processor with embedded programmable logic. In *Neural information processing systems conference (NIPS)*, 2013.

[12] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11):1–4, 2015.

[13] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[14] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of*

the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pages 449–460. IEEE Computer Society, 2012.

[15] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. In *ISCAS*, volume 2010, pages 257–260, 2010.

[16] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 682–687, 2014.

[17] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1-3):239–255, 2010.

[18] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11):1–4, 2015.

[19] M Vardhana, N Arunkumar, Sunitha Lasrado, Enas Abdulhay, and Gustavo Ramirez-Gonzalez. Convolutional neural network for bio-medical image segmentation with hardware acceleration. *Cognitive Systems Research*, 50:10–14, 2018.

[20] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. On-device neural net inference with mobile gpus. *arXiv preprint arXiv:1907.01989*, 2019.

[21] Pengfei Zhu, Longyin Wen, Xiao Bian, Haibin Ling, and Qinghua Hu. Vision meets drones: A challenge. *arXiv preprint arXiv:1804.07437*, 2018.

[22] Geesara Prathap and Ilya Afanasyev. Deep learning approach for building detection in satellite multispectral imagery. In *2018 International Conference on Intelligent Systems (IS)*, pages 461–465. IEEE, 2018.

[23] Michal Kawulok, Szymon Piechaczek, Krzysztof Hrynczenko, Pawel Benecki, Daniel Kostrzewa, and Jakub Nalepa. On training deep networks for satellite image super-resolution. *arXiv preprint arXiv:1906.06697*, 2019.

[24] Donald M Hassler, Cary Zeitlin, Robert F Wimmer-Schweingruber, Bent Ehresmann, Scot Rafkin, Jennifer L Eigenbrode, David E Brinza, Gerald Weigle, Stephan Böttcher, Eckart Böhm, et al. Marsâ surface radiation environment measured with the mars science laboratoryâs curiosity rover. *science*, 343(6169):1244797, 2014.

[25] LA Leshin, PR Mahaffy, CR Webster, Michel Cabane, Patrice Coll, PG Conrad, PD Archer, SK Atreya, AE Brunner, A Buch, et al. Volatile, isotope, and organic analysis of martian fines with the mars curiosity rover. *Science*, 341(6153):1238937, 2013.

[26] Richard Welch, Daniel Limonadi, and Robert Manning. Systems engineering the curiosity rover: A retrospective. In *2013 8th International Conference on System of Systems Engineering*, pages 70–75. IEEE, 2013.

[27] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[28] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *arXiv preprint arXiv:1604.04494*, 2016.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[30] Yunfeng Zhang, Qinglan Fan, Fangxun Bao, Yifang Liu, and Caiming Zhang. Single-image super-resolution based on rational fractal interpolation. *IEEE Transactions on Image Processing*, 27(8):3782–3797, 2018.

[31] Tianwei Lin, Xu Zhao, and Zheng Shou. Single shot temporal action detection. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 988–996. ACM, 2017.

[32] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese cnn for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40, 2016.

[33] Hanxi Li, Yi Li, and Fatih Porikli. Deeptrack: Learning discriminative feature representations online for robust visual tracking. *IEEE Transactions on Image Processing*, 25(4):1834–1848, 2015.

[34] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[35] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3676–3684, 2015.

[36] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

[37] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, volume 2, page 6, 2017.

[38] Laura Sevilla-Lara, Yiyi Liao, Fatma Guney, Varun Jampani, Andreas Geiger, and Michael J Black. On the integration of optical flow and action recognition. *arXiv preprint arXiv:1712.08416*, 2017.

[39] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199. Springer, 2014.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[41] Louis Sokoloff. The metabolism of the central nervous system in vivo. *Handbook of physiology, section I, neurophysiology*, 3:1843–1864, 1960.

[42] Lois Sokoloff. Circulation and energy metabolism of the brain. *Basic neurochemistry*, 2:338–413, 1989.

[43] DF Rolfe and Guy C Brown. Cellular energy utilization and molecular origin of standard metabolic rate in mammals. *Physiological reviews*, 77(3):731–758, 1997.

[44] Graeme Mitchison. Axonal trees and cortical architecture. *Trends in neurosciences*, 15(4):122–126, 1992.

[45] Alexei A Koulakov and Dmitri B Chklovskii. Orientation preference patterns in mammalian visual cortex: a wire length minimization approach. *Neuron*, 29(2):519–527, 2001.

[46] William B Levy and Robert A Baxter. Energy efficient neural codes. *Neural computation*, 8(3):531–543, 1996.

[47] Roland Baddeley, Larry F Abbott, Michael CA Booth, Frank Sengpiel, Tobe Freeman, Edward A Wakeman, and Edmund T Rolls. Responses of neu-

rons in primary and inferior temporal visual cortices to natural scenes. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 264(1389):1775–1783, 1997.

[48] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191, 2017.

[49] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[50] Carissa Lew Steve Branson Alexander G. Anderson Lubomir Bourdev Oren Rippel, Sanjay Nair. Learned video compression. 2018.

[51] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2391–2400. JMLR. org, 2017.

[52] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402, 2018.

[53] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[54] Christopher M Bishop, Andrew Blake, and Bhaskara Marthi. Super-resolution enhancement of video. In *AISTATS*, 2003.

[55] Mohammad Moinul Islam, Vijayan K Asari, Mohammed Nazrul Islam, and Mohammad A Karim. Super-resolution enhancement technique for low res-

olution video. *IEEE Transactions on Consumer Electronics*, 56(2):919–924, 2010.

[56] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[57] Cisco Visual Networking Index. The zettabyte era - Trends and analysis. *Cisco White Paper*, 2016.

[58] Zhongwen Xu, Yi Yang, and Alex G Hauptmann. A discriminative cnn video representation for event detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1798–1807, 2015.

[59] Zhen Dong, Su Jia, Tianfu Wu, and Mingtao Pei. Face video retrieval via deep learning of binary hash representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[60] Khan Muhammad, Jamil Ahmad, Zhihan Lv, Paolo Bellavista, Po Yang, and Sung Wook Baik. Efficient deep cnn-based fire detection and localization in video surveillance applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(7):1419–1434, 2018.

[61] Cheng-Bin Jin, Shengzhe Li, Trung Dung Do, and Hakil Kim. Real-time human action recognition using cnn over temporal images for static video surveillance cameras. In *Pacific Rim Conference on Multimedia*, pages 330–339. Springer, 2015.

[62] Advanced video coding for generic audio-visual services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC). ITU-T and ISO/IEC JTC 1, 2003.

[63] High efficient video coding for generic audio-visual services, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC). ITU-T and ISO/IEC JTC 1/SC 29/WG 11, 2015. and subsequent editions.

[64] W. J. Han G. J. Sullivan, J. R. Ohm and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circ. and Syst. for Vid. Technol.*, 22(12):1649–1668, Dec 2012.

[65] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[66] Litong Feng Wei Zhang Zhaoyang Zhang, Zhanghui Kuang. Temporal sequence distillation: Towards few-frame action recognition in videos. In *Arxiv: 1808.05085*, 2018.

[67] Bowen Zhang, Limin Wang, Zhe Wang, Yu Qiao, and Hanli Wang. Real-time action recognition with deeply transferred motion vector cnns. *IEEE Transactions on Image Processing*, 27(5):2326–2339, 2018.

[68] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[69] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[70] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[71] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[72] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[73] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[74] Donald F Specht. Probabilistic neural networks. *Neural networks*, 3(1):109–118, 1990.

[75] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[76] Baidu Co. Convolutional neural networks for image classification: http://book.paddlepaddle.org/03.image_classification. 2015.

[77] Tourapis Alexis, Sullivan Gary, Suhring Karsten, Oelbaum Tobias, and Leontaris Athanasios. H.264 reference software. *http://iphome.hhi.de/suehring/tml/*, 2009.

[78] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[80] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech*, volume 11, pages 73–5, 2013.

[81] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems*, pages 190–198, 2013.

[82] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[83] Sune Bao. Talking about convolutional neural networks: https://ireneli.eu/2016/02/03/deep-learning-05-talking-convolutional-neural-networks/. 2014.

[84] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[85] Elizabeth Gardner. Maximum storage capacity in neural networks. *EPL (Europhysics Letters)*, 4(4):481, 1987.

[86] William A Little and Gordon L Shaw. Analytic study of the memory storage capacity of a neural network. *Mathematical biosciences*, 39(3-4):281–290, 1978.

[87] Leon O Chua and Tamas Roska. The cnn paradigm. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(3):147–156, 1993.

[88] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[89] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[90] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[91] Guoliang Kang, Jun Li, and Dacheng Tao. Shakeout: A new regularized deep neural network training scheme. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[92] Nicolas Le Roux and Yoshua Bengio. Continuous neural networks. In *Artificial Intelligence and Statistics*, pages 404–411, 2007.

[93] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[94] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[95] Mee Young Park and Trevor Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007.

[96] Rudy Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural computation*, 9(1):185–204, 1997.

[97] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[98] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[99] Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in neural information processing systems*, pages 2814–2822, 2013.

[100] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[101] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[102] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[103] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[104] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[105] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[106] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661. IEEE, 2016.

[107] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[108] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[109] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015.

[110] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *International conference on artificial neural networks*, pages 281–290. Springer, 2014.

[111] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[112] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[113] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[114] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[115] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[116] Jianlong Fu, Heliang Zheng, and Tao Mei. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4438–4446, 2017.

[117] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[118] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[119] Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 245–249. IEEE, 2014.

[120] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[121] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[122] A Sakar and Richard J. Mammone. Growing and pruning neural tree networks. *IEEE Transactions on Computers*, 42(3):291–299, 1993.

[123] Raul Vicen-Bueno, Roberto Gil-Pita, Maria P Jarabo-Amores, and Francisco López-Ferreras. Complexity reduction in neural networks applied to traffic sign recognition tasks. In *2005 13th European Signal Processing Conference*, pages 1–4. IEEE, 2005.

[124] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[125] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *arXiv preprint arXiv:1704.02654*, 2017.

[126] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[127] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM, 2014.

[128] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[129] Wai Keung Wong, Zhihui Lai, Jiajun Wen, Xiaozhao Fang, and Yuwu Lu. Low-rank embedding for robust image feature extraction. *IEEE Transactions on Image Processing*, 26(6):2905–2917, 2017.

[130] Yue Li, Dong Liu, Houqiang Li, Li Li, Zhu Li, and Feng Wu. Learning a convolutional neural network for image compact-resolution. *IEEE Transactions on Image Processing*, 28(3):1092–1107, 2019.

[131] Seong-Ping Chuah, Ngai-Man Cheung, and Chau Yuen. Layered coding for mobile cloud gaming using scalable blinn-phong lighting. *IEEE Transactions on Image Processing*, 25(7):3112–3125, 2016.

[132] Wenjie Pei, Tadas Baltrusaitis, David MJ Tax, and Louis-Philippe Morency. Temporal attention-gated model for robust sequence classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6730–6739, 2017.

[133] Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Wei Liu, and Tat-Seng Chua. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5659–5667, 2017.

[134] Wenguan Wang, Jianbing Shen, and Ling Shao. Video salient object detection via fully convolutional networks. *IEEE Transactions on Image Processing*, 27(1):38–49, 2018.

[135] Hao Chen and Youfu Li. Three-stream attention-aware network for rgb-d salient object detection. *IEEE Transactions on Image Processing*, 2019.

[136] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[137] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[138] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[139] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[140] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[141] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[142] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[143] Lei Xu, Michael I Jordan, and Geoffrey E Hinton. An alternative model for mixtures of experts. In *Advances in neural information processing systems*, pages 633–640, 1995.

[144] Erik Bochinski, Rolf Jongebloed, Michael Tok, and Thomas Sikora. Regularized gradient descent training of steered mixture of experts for sparse image representation. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 3873–3877. IEEE, 2018.

[145] Christoph Bregler and Jitendra Malik. Learning appearance based models: Mixtures of second moment experts. In *Advances in Neural Information Processing Systems*, pages 845–851, 1997.

[146] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017.

[147] Lieven Lange, Ruben Verhack, and Thomas Sikora. Video representation and coding using a sparse steered mixture-of-experts network. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2016.

[148] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, pages 827–832, 2015.

[149] Alan E Gelfand, Susan E Hills, Amy Racine-Poon, and Adrian FM Smith. Illustration of bayesian inference in normal data models using gibbs sampling. *Journal of the American Statistical Association*, 85(412):972–985, 1990.

[150] Christopher M Bishop and Markus Svenskn. Bayesian hierarchical mixtures of experts. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 57–64. Morgan Kaufmann Publishers Inc., 2002.

[151] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[152] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[153] Steven J Nowlan and Geoffrey E Hinton. Evaluation of adaptive mixtures of competing experts. In *Advances in neural information processing systems*, pages 774–780, 1991.

[154] Steve R Waterhouse, David MacKay, and Anthony J Robinson. Bayesian methods for mixtures of experts. In *Advances in neural information processing systems*, pages 351–357, 1996.

[155] Ruben Verhack, Thomas Sikora, Lieven Lange, Glenn Van Wallendael, and Peter Lambert. A universal image coding approach using sparse steered mixture-of-experts regression. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 2142–2146. IEEE, 2016.

[156] Xin Wang, Jiawei Wu, Da Zhang, Yu Su, and William Yang Wang. Learning to compose topic-aware mixture of experts for zero-shot video captioning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8965–8972, 2019.

[157] Lei Xu, Michael I Jordan, and Geoffrey E Hinton. An alternative model for mixtures of experts. In *Advances in neural information processing systems*, pages 633–640, 1995.

[158] Zhuobin Zheng, Chun Yuan, Xinrui Zhu, Zhihui Lin, Yangyang Cheng, Cheng Shi, and Jiahui Ye. Self-supervised mixture-of-experts by uncertainty

estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5933–5940, 2019.

[159] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[160] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[161] Yan Zhou and Murat Kantarcioglu. Adversarial learning with bayesian hierarchical mixtures of experts. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 929–937. SIAM, 2014.

[162] Ran Avnimelech and Nathan Intrator. Boosted mixture of experts: an ensemble learning scheme. *Neural computation*, 11(2):483–497, 1999.

[163] David W Opitz and Jude W Shavlik. Actively searching for an effective neural network ensemble. *Connection Science*, 8(3-4):337–354, 1996.

[164] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

[165] Ronan Collobert, Yoshua Bengio, and Samy Bengio. Scaling large learning problems with hard parallel mixtures. *International Journal of pattern recognition and artificial intelligence*, 17(03):349–365, 2003.

[166] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[167] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[168] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3214–3223, 2018.

[169] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems*, pages 1141–1151, 2017.

[170] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[171] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems*, pages 10771–10780, 2018.

[172] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[173] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[174] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[175] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using aï¿Œ laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[176] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[177] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10629–10638, 2019.

[178] John Miano. *Compressed image file formats: Jpeg, png, gif, xbm, bmp*. Addison-Wesley Professional, 1999.

[179] Greg Roelofs and Richard Koman. *PNG: the definitive guide*. O'Reilly & Associates, Inc., 1999.

[180] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176, 2017.

[181] Swami Sankaranarayanan, Yogesh Balaji, Carlos D Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2018.

[182] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, pages 1640–1650, 2018.

[183] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[184] Nal Kalchbrenner, L Espeholt, O Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in Neural Information Processing Systems*, 2016.

[185] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster r-cnn architecture

for temporal action localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1130–1139, 2018.

[186] Jingjia Huang, Nannan Li, Tao Zhang, Ge Li, Tiejun Huang, and Wen Gao. Sap: Self-adaptive proposal model for temporal action detection based on reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[187] Dong Li, Zhaofan Qiu, Qi Dai, Ting Yao, and Tao Mei. Recurrent tubelet proposal and recognition networks for action detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 303–318, 2018.

[188] Xitong Yang, Xiaodong Yang, Ming-Yu Liu, Fanyi Xiao, Larry S Davis, and Jan Kautz. Step: Spatio-temporal progressive learning for video action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 264–272, 2019.

[189] Yubo Zhang, Pavel Tokmakov, Martial Hebert, and Cordelia Schmid. A structured model for action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9975–9984, 2019.

[190] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[191] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[192] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer, 2004.

[193] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4489–4497, 2015.

[194] Vadim Kantorov and Ivan Laptev. Efficient feature extraction, encoding and classification for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2593–2600, 2014.

[195] Bowen Zhang, Limin Wang, Zhe Wang, Yu Qiao, and Hanli Wang. Real-time action recognition with enhanced motion vector cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2718–2726, 2016.

[196] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. Knowl. and Data Eng.*, 22(10):1345–1359, Oct 2010.

[197] Chaoyuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 6026–6035, 2018.

[198] Ke Xu, Xinghao Jiang, and Tanfeng Sun. Two-stream dictionary learning architecture for action recognition. *IEEE Trans. on Circ. and Syst. for Video Technol.*, 27(3):567–576, Feb 2017.

[199] Shichao Zhao, Yanbin Liu, Yahong Han, Richang Hong, Qinghua Hu, and Qi Tian. Pooling the convolutional layers in deep convnets for video action recognition. *IEEE Trans. on Circ. and Syst. for Video Technol.*, 28:1839–1849, Mar 2017.

[200] Rohit Girdhar, Deva Ramanan, Abhinav Gupta, Josef Sivic, and Bryan Russell. ActionVLAD: Learning spatio-temporal aggregation for action classification. *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3165–3174, 2017.

[201] Jorge Posada, Carlos Toro, Iñigo Barandiaran, David Oyarzun, Didier Stricker, Raffaele Amicis, Eduardo B Pinto, Peter Eisert, Jürgen Döllner, and Ivan Vallarino. Visual computing as a key enabling technology for industrie 4.0 and industrial internet. *IEEE Trans. Computer Graphics and Applications*, 35(2):26–40, Mar 2015.

[202] Alessandro Redondi, Matteo Cesana, and Marco Tagliasacchi. Rate-accuracy optimization in visual wireless sensor networks. *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pages 1105–1108, 2012.

[203] Alessandro Redondi, Luca Baroffio, Matteo Cesana, and Marco Tagliasacchi. Compress-then-analyze vs. analyze-then-compress: Two paradigms for image analysis in visual sensor networks. *presented at the 15th Int. Workshop on Multimedia Signal Processing (MMSP)*, pages 278–282, 2013.

[204] Rohit Girdhar and Deva Ramanan. Attentional pooling for action recognition. *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 33–44, 2017.

[205] Alhabib Abbas, Mohammad Jubran, Aaron Chadha, and Yiannis Andreopoulos. Rate-accuracy trade-off in video classification with deep convolutional neural networks. *Proc. IEEE Int. Conf. on Image Processing (ICIP)*, pages 793–797, 2018.

[206] Miguel T Coimbra and Mike Davies. Approximating optical flow within the MPEG-2 compressed domain. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(1):103–107, 2005.

[207] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[208] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[209] FFMPEG/LibAV open-source codec and API documentation. *http://ffmpeg.org/libavcodec.html*.

[210] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2556–2563. IEEE, 2011.

[211] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[212] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[213] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[214] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv., DOI:10.1145/356924.356930*, 16(2):187–260, Jun 1984.

[215] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[216] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[217] Kyoung Kwon, Yin Shen, and Jay Kuo. Rate control for H.264 video with enhanced rate and distortion models. *IEEE Trans. Circuits and Syst. for Video Technol.*, 17(5):517–529, May 2007.

[218] Morris H DeGroot and Mark J Schervish. *Probability and statistics*. Pearson Education, 2012.

[219] A Conn, Nick Gould, and Ph Toint. A globally convergent lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Mathematics of Computation of the American Mathematical Society*, 66(217):261–288, 1997.

[220] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

[221] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 4724–4733. IEEE, 2017.

[222] Siddharth Srivastava and Brejesh Lall. Superresolution based medical image compression for mobile platforms. In *Workshop on Machine Learning for HealthCare*, 2015.

[223] Jim Martin, Yunhui Fu, Nicholas Wourms, and Terry Shaw. Characterizing netflix bandwidth consumption. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 230–235. IEEE, 2013.

[224] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[225] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[226] Mohammad Jubran, Alhabib Abbas, Aaron Chadha, and Yiannis Andreopoulos. Rate-accuracy trade-off in video classification with deep convolutional neural networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[227] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European Conference on Computer Vision*, pages 391–407. Springer, 2016.

[228] Karl Heinz Borgwardt. *The Simplex Method: a probabilistic analysis*, volume 1. Springer Science & Business Media, 2012.

[229] Laurent Sifre and Stéphane Mallat. Rigid-motion scattering for texture classification. *arXiv preprint arXiv:1403.1687*, 2014.

[230] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[231] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[232] Xu Tang, Daniel K Du, Zeqiang He, and Jingtuo Liu. Pyramidbox: A context-assisted single shot face detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 797–813, 2018.

[233] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. S3fd: Single shot scale-invariant face detector. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 192–201, 2017.

[234] Zhishuai Zhang, Siyuan Qiao, Cihang Xie, Wei Shen, Bo Wang, and Alan L Yuille. Single-shot object detection with enriched semantics. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5813–5821, 2018.

[235] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9259–9266, 2019.

[236] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[237] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[238] Shangqi Gao and Xiahai Zhuang. Multi-scale deep neural networks for real image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[239] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *2009 IEEE 12th international conference on computer vision*, pages 349–356. IEEE, 2009.

[240] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei Zhang. Ntire 2017 challenge on single image super-resolution: Methods and results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 114–125, 2017.