PRIVACY PRESERVING AND EFFICIENT MACHINE LEARNING ALGORITHMS

by

Efstathia Soufleri

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2024

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Kaushik Roy, Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Vijay Raghunathan

School of Electrical and Computer Engineering

Dr. Sumeet Gupta

School of Electrical and Computer Engineering

Approved by:

Dr. Milind Kulkarni

This thesis is dedicated to my beloved family, friends, and teachers

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to all the amazing people that I have had the privilege of working with over the past years.

First and foremost, I express my immense gratitude to my PhD advisor, Professor Kaushik Roy, for his research insights and advice that helped me complete this work. He has patiently taught me how to approach and solve complex research problems. He instilled in me self-confidence and advised me about both academic and non-academic issues.

I would also like to thank the members of my advisory committee: Prof. Anand Raghunathan, Prof. Vijay Raghunathan, and Prof. Sumeet Gupta. Their constructive feedback and valuable insights have been crucial in refining my research and thesis. I am genuinely thankful for their dedication and the time that they have devoted to my academic pursuits.

Furthermore, I am deeply grateful to Prof. George Stamoulis from the University of Thessaly. His encouragement and mentorship were essential in motivating me to pursue further studies in the United States. His expertise and guidance have left a lasting impact on my academic and personal growth.

I also want to thank the members of the Nanoelectronics Research Laboratory (NRL), who have enriched my experience with countless engaging discussions. I am particularly thankful to Deepak Ravikumar, Tanvi Sharma, Sarada Krithivasan, Isha Garg, Manish Nagaraj, Timur Ibrayev, Wachirawit Ponghiran, Sangamesh Kodge, Adarsh Kosta, Utkarsh Saxena, Aparna Aketi, Sakshi Choudhary, Marco Apolinario, Shristi Das Biswas, Soumyadeep Chandra, Sayeed Shafayet Chowdhury, Jimmy Gammell, Kang He, Amogh Joshi, Gaurav Kumar, Dong Eun Kim, Shubham Negi, Yinghan Long, Deepika Sharma, and Eunseon Yu for making the lab an inspiring and collaborative environment. I am also grateful to the NRL alumni, including Mei-Chin Chen, Maryam Parsa, Gopalakrishnan Srinivasan, Akhilesh Jaiswal, Robert Andrawis, Aayush Anikt, Amogh Agrawal, Chankyu Lee, Indranil Chakraborty, Nitin Rathi, Deboleena Roy, Saima Sharmin, and Chamika Liyanagedera. Additionally, I extend my thanks to Nicole Piegza for her assistance in navigating various administrative and technical challenges.

Beyond the academic sphere, I am grateful for the support and encouragement of my family. My parents, Konstantinos and Evangelia, and my sister, Dimitra, have been my pillars of strength. Their belief in my abilities has provided me with the foundation to pursue my dreams, and I am eternally grateful for their love and support.

As I embark on the next phase of my journey, I carry with me the lessons learned and the relationships forged during my PhD. I appreciate the opportunities that have shaped me into who I am today, and I look forward to continued growth and learning in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Extensive data availability has catalyzed the expansion of deep learning. Such advancements include image classification, speech, and natural language processing. However, this data-driven progress is often hindered by privacy restrictions preventing the public release of specific datasets. For example, some vision datasets cannot be shared due to privacy regulations, particularly those containing images depicting visually sensitive or disturbing content. At the same time, it is imperative to deploy deep learning efficiently, specifically Deep Neural Networks (DNNs), which are the core of deep learning. In this dissertation, we focus on achieving efficiency by reducing the computational cost of DNNs in multiple ways.

This thesis first tackles the privacy concerns arising from deep learning. It introduces a novel methodology that synthesizes and releases synthetic data, instead of private data. Specifically, we propose Differentially Private Image Synthesis (DP-ImgSyn) for generating and releasing synthetic images used for image classification tasks. These synthetic images satisfy the following three properties: (1) they have DP guarantees, (2) they preserve the utility of private images, ensuring that models trained using synthetic images result in comparable accuracy to those trained on private data, and (3) they are visually dissimilar from private images. The DP-ImgSyn framework consists of the following steps: firstly, a teacher model is trained on private images using a DP training algorithm. Subsequently, public images are used for initializing synthetic images, which are optimized in order to be aligned with the private dataset. This optimization leverages the teacher network's batch normalization layer statistics (mean, standard deviation) to inject information from the private dataset into the synthetic images. Third, the synthetic images and their soft labels obtained from the teacher model are released and can be employed for neural network training in image classification tasks.

As a second direction, this thesis delves into achieving efficiency in deep learning. With neural networks widely deployed for tackling diverse and complex problems, the resulting models often become parameter-heavy, demanding substantial computational resources for deployment. To address this challenge, we focus on quantizing the weights and the activations of DNNs. In more detail, we propose a method for compressing neural networks

through layer-wise mixed-precision quantization. Determining the optimal bit widths for each layer is a non-trivial task, given the fact that the search space is exponential. Thus, we employ a Multi-Layer Perceptron (MLP) trained to determine the suitable bit-width for each layer. The Kullback-Leibler (KL) divergence of softmax outputs between the quantized and full precision networks is the metric used to gauge quantization quality. We experimentally investigate the relationship between KL divergence and network size, noting that more aggressive quantization correlates with higher divergence and vice versa. The MLP is trained using the layer-wise bit widths as labels and their corresponding KL divergence as inputs. To generate the training set, pairs of layer-wise bit widths and their respective KL divergence values are obtained through Monte Carlo sampling of the search space. This approach aims to reduce the computational cost of DNN deployment, while maintaining high classification accuracy.

Additionally, we aim to enhance efficiency in machine learning by introducing a computationally efficient method for action recognition on compressed videos. Rather than decompressing videos for action recognition tasks, our approach performs action recognition directly on the compressed videos. This is achieved by leveraging the modalities within the compressed video format, specifically motion vectors, residuals, and intra-frames. To process each modality, we deploy three neural networks. Our observations indicate a hierarchy in convergence behavior: the network processing intra-frames tend to converge to a flatter minimum than the network processing residuals, which, in turn, converge to a flatter minimum than the motion vector network. This hierarchy motivates our strategy for knowledge transfer among modalities to achieve flatter minima, generally associated with better generalization. Based on this insight, we propose Progressive Knowledge Distillation (PKD), a technique that incrementally transfers knowledge across modalities. This method involves attaching early exits, known as Internal Classifiers (ICs), to the three networks. PKD begins by distilling knowledge from the motion vector network, then the residual network, and finally the intra-frame network, sequentially improving the accuracy of the ICs. Moreover, we introduce Weighted Inference with Scaled Ensemble (WISE), which combines outputs from the ICs using learned weights, thereby boosting accuracy during inference. The combination

of PKD and WISE demonstrates significant improvements in efficiency and accuracy for action recognition on compressed videos.

In summary, this dissertation contributes to advancing privacy preserving and efficient machine learning algorithms. The proposed methodologies offer practical solutions for deploying machine learning systems in real-world scenarios by addressing data privacy and computational efficiency. Through innovative approaches to image synthesis, neural network compression, and action recognition, this work aims to foster the development of robust and scalable machine learning frameworks for diverse computer vision applications.

1. INTRODUCTION

1.1 Challenges of Achieving Privacy in Machine Learning

In recent years, deep learning has driven major innovations across various fields, including healthcare, finance, autonomous vehicles, and personalized recommendations. These advances are due to deep learning models' ability to extract patterns from large amounts of data. However, this heavy reliance on data raises significant privacy concerns.

To address these privacy issues, various regulations have been established. For example, the European Union General Data Protection Regulation (GDPR) [1], the Health Insurance Portability and Accountability Act (HIPAA) [2], and the California Consumer Privacy Act (CCPA) [3] aim to protect personal data. They establish regulations that require organizations to be transparent and accountable in their data processing practices. This poses challenges when trying to apply machine learning using these data.

Specifically, the GDPR imposes rules on collecting, storing, and processing personal data, requiring a careful balance between innovation and compliance. Similarly, HIPAA protects sensitive patient health information, including electronic health records. CCPA gives consumers rights over the data held by businesses, such as knowing what information is collected and requesting its deletion.

Privacy and machine learning involve ethical, legal, and technical considerations. Ethically, the responsible use of personal data is essential, especially as machine learning algorithms increasingly influence various aspects of society. Without adequate safeguards, the use of personal data can raise concerns about fairness, transparency, and accountability.

Legally, privacy in machine learning is governed by complex regulations and compliance requirements. Organizations need to adhere to principles of ethical data collection, purpose limitation, and transparency in their data processing. The scope of regulations like the GDPR extends these requirements to organizations outside the EU, complicating compliance for multinational companies.

Technically, privacy in machine learning poses challenges related to data anonymization, encryption, and secure computation. Traditional anonymization methods might not provide sufficient protection, as adversaries can re-identify [4], [5] individuals using auxiliary infor-

mation. Balancing data utility and privacy preservation requires innovative approaches, such as federated learning [6], [7], differential privacy [8], and homomorphic encryption [9], [10].

Federated learning enables collaborative model training across distributed datasets without centralized data aggregation, reducing privacy risks. Differential privacy adds noise to query responses to protect sensitive information while preserving aggregate statistics. Homomorphic encryption allows secure computation on encrypted data, enabling models to operate without exposing raw data to unauthorized parties.

Given these challenges, synthetic data generation has emerged as a promising solution to balance data utility and privacy protection. Synthetic data, generated through techniques such as generative adversarial networks (GANs) [11], variational autoencoders (VAEs) [12], and differential privacy [8], try to have the utility of real-world data while preserving privacy.

However, synthetic data generation also has its limitations. Ensuring the utility of synthetic data requires careful consideration. Ethical concerns should be addressed to prevent unintended consequences or misuse.

Overall, privacy is an important aspect of developing and deploying machine learning algorithms, especially with stringent regulatory requirements and growing privacy concerns. Embracing privacy as a fundamental principle in machine learning ensures that the benefits of data-driven decision-making are realized responsibly and ethically.

1.2 Challenges of Achieving Efficiency in Machine Learning

Efficiency in Machine Learning (ML) encompasses various aspects, including computational resources, memory usage, and model performance. As ML models become increasingly complex, with neural networks featuring 100's billions of parameters, achieving efficiency becomes paramount. This dissertation explores the challenges faced in achieving efficiency in ML, focusing on two key aspects: parameter optimization and data processing. The discussion is motivated by the need to address the computational demands of large neural networks and the efficiency gains achievable through optimized data processing techniques.

Neural networks, particularly deep learning models, exhibit a high degree of complexity due to their numerous parameters and layers. Each parameter contributes to the model's

ability to capture intricate patterns in the data, but it also increases computational overhead. The sheer size of neural networks poses challenges in terms of memory requirements, training time, and inference speed. Additionally, the process of training these models involves iterative optimization algorithms that require significant computational resources. Thus, optimizing the parameters and reducing the computational complexity of neural networks are essential steps in achieving efficiency.

To address the challenges posed by the large number of parameters in neural networks, various optimization techniques have been proposed. One approach is weight quantization [13], which involves reducing the precision of weight values to reduce memory usage and computational complexity. By quantizing weights to lower bit precision, such as 8-bit or even binary values, significant reductions in memory footprint and computational overhead can be achieved. However, weight quantization must be performed carefully to minimize the impact on model accuracy.

In addition to weight quantization, activation quantization [13], [14] is another strategy for improving the efficiency of neural networks. Activation functions introduce non-linearity into neural network architectures, but they also contribute to computational complexity. By quantizing activation values, researchers can reduce the memory footprint and computational requirements of neural network inference. Techniques such as dynamic quantization and uniform quantization have been proposed to quantize activation values effectively while preserving model accuracy.

While quantization techniques offer promising avenues for improving efficiency in neural networks, they are not without challenges. One challenge is balancing the trade-off between model accuracy and quantization-induced error. Aggressive quantization may lead to significant accuracy degradation, particularly in complex tasks such as image recognition or natural language processing. Thus, researchers must carefully tune quantization parameters to minimize the impact on model performance while maximizing efficiency gains.

Another aspect of efficiency in ML involves optimizing data processing pipelines to minimize computational overhead. Raw data, such as uncompressed videos, can be computationally intensive to process, requiring decompression before analysis. However, leveraging directly compressed data can offer significant efficiency gains. Compressed video formats,

such as H.264 or H.265, encode video data in a compressed format, reducing the amount of data that needs to be processed during inference.

Directly using compressed video data in ML applications offers several advantages in terms of efficiency. Firstly, it reduces the computational overhead associated with decompression, allowing for faster processing and inference. Secondly, compressed data requires less storage space, leading to lower memory requirements and reduced I/O overhead. Additionally, leveraging compressed data can facilitate real-time processing and analysis of video streams, enabling applications such as video surveillance, video analytics, and live streaming.

While using compressed data offers efficiency gains, it also presents challenges and considerations. One challenge is ensuring compatibility with ML models, as compressed data may require specialized decoding techniques or preprocessing steps. Moreover, compressed data may introduce artifacts or information loss, potentially impacting the performance of ML algorithms. Thus, researchers must carefully evaluate the trade-offs between efficiency and data quality when leveraging compressed data in ML applications.

Efficiency is a critical consideration in ML, particularly as models become increasingly complex and train on large scale datasets. Addressing the challenges of parameter optimization and data processing is essential for achieving efficiency gains in neural networks. Techniques such as weight and activation quantization offer promising approaches for reducing computational complexity while preserving model accuracy. Similarly, leveraging directly compressed data can improve efficiency by minimizing the computational overhead associated with data processing. Future research directions include exploring advanced quantization techniques, optimizing data processing pipelines, and developing specialized hardware accelerators to further enhance efficiency in ML applications. By addressing these challenges and leveraging innovative solutions, we can unlock the full potential of ML while maximizing efficiency and scalability.

1.3 Dissertation Contributions

We begin by highlighting the contributions of this dissertation, which focuses on both privacy and efficiency in machine learning. For privacy, one aspect of this dissertation

introduces a method for releasing synthetic data to protect privacy. By using synthetic data instead of private data, individuals or organizations can maintain confidentiality while still using data to train neural networks for various tasks. Our approach allows the integration and use of data from different sources to build strong neural models.

Shifting our focus to efficiency in machine learning, we initially worked on improving inference efficiency. This is achieved through neural model compression, where both weights and activations are quantized to reduce accuracy loss. We introduce mixed precision quantization, where different bit precisions are assigned to different layers of the network based on their importance. Using a multi-layer perceptron (MLP), we dynamically allocate bit widths across network layers. We also explore efficiency gains through action recognition on compressed videos. In this case, classification tasks are performed directly on compressed videos, removing the need for video decompression before model input. Additionally, our framework includes early exits, allowing some video instances to be classified using only certain compressed video modalities, further enhancing efficiency.

The following paragraphs explain the main findings of each effort in these areas.

1.3.1 Privacy Preserving Machine Learning

The proliferation of vast datasets has significantly advanced deep learning algorithms for visual tasks. However, privacy concerns often render some visual datasets inaccessible. To address this challenge, substituting real images with synthetic ones has become a common approach. Generative Adversarial Networks (GANs) incorporating Differential Privacy (DP) assurances are frequently used for this purpose. However, synthetic images produced by GANs often closely resemble private ones, which can be problematic, especially when the private data contains visually sensitive or disturbing content.

To mitigate this issue, we propose a novel framework called Differentially Private Image Synthesis (DP-ImgSyn), which generates synthetic images suitable for image classification tasks without relying on generative methods. The synthetic images generated by DP-ImgSyn offer three key advantages: (1) they come with DP guarantees, (2) they preserve the utility

of private images, ensuring that models trained on synthetic data achieve similar accuracy to those trained on private data, and (3) they are visually distinct from private images.

DP-ImgSyn operates through the following steps: First, a teacher model is trained on private images using a DP training algorithm. Second, public images serve as a starting point for generating synthetic images, which are then optimized to align with the private images. This optimization process leverages the batch normalization layer statistics (mean and standard deviation) of the teacher network to incorporate information from the private images into the synthetic ones. Finally, the synthetic images, along with their soft labels obtained from the teacher model, are released for deployment in neural network training for image classification tasks.

Experimental results across various image classification datasets demonstrate that our framework outperforms generative techniques, achieving up to approximately a 20% increase in image classification accuracy when employing similar DP training mechanisms.

1.3.2 Neural Network Compression via Layer-wise Quantization

We propose an innovative method for assigning layer-wise bit widths in a neural network using a multi-layer perceptron (MLP). The Kullback-Leibler (KL) divergence of the softmax outputs between the quantized network and the full precision network is utilized as a metric to measure the quality of quantization. We investigate the relationship between KL divergence and network size, and our experiments reveal that more aggressive quantization results in higher divergence, while less aggressive quantization leads to lower divergence.

The MLP is trained using layer-wise bit widths as labels and their corresponding KL divergence as input. The training set, which consists of pairs of layer-wise bit widths and their associated KL divergence, is gathered using Monte Carlo sampling of the exponential search space. To ensure the MLP learns to predict bit widths that minimize network size, we introduce a penalty term in the loss function.

Our results show that the bit-width predictions from the trained MLP lead to a reduced network size without compromising accuracy, achieving better or comparable results to state-of-the-art methods with less computational overhead. Specifically, our method achieves up

to 6x, 4x, and 4x compression on VGG16, ResNet50, and GoogLeNet, respectively, with no loss in accuracy compared to the original full precision pre-trained models on the ImageNet dataset.

1.3.3 Efficient Action Recognition on Compressed Videos

Compressed video action recognition classifies video samples by utilizing the different modalities present in compressed videos, such as motion vectors, residuals, and intra-frames. To achieve this, we deploy three neural networks, each specialized in processing one modality. Our observations show that the network processing intra-frames converge to a flatter minimum compared to the network processing residuals, which in turn converge to a flatter minimum than the motion vector network. This convergence hierarchy guides our knowledge transfer strategy among modalities to achieve flatter minima, which are generally linked to better generalization.

With this understanding, we introduce Progressive Knowledge Distillation (PKD), a technique that incrementally transfers knowledge across the modalities. This method involves attaching early exits, known as Internal Classifiers (ICs), to the three networks. PKD starts by distilling knowledge from the motion vector network, then moves to the residual network, and finally to the intra-frame network, sequentially enhancing IC accuracy. Furthermore, we propose Weighted Inference with Scaled Ensemble (WISE), which combines outputs from the ICs using learned weights, thereby boosting accuracy during inference.

Our experiments demonstrate the effectiveness of training the ICs with PKD compared to standard cross-entropy-based training, showing IC accuracy improvements of up to 5.87% and 11.42% on the UCF-101 and HMDB-51 datasets, respectively. Additionally, WISE improves accuracy by up to 4.28% and 9.30% on UCF-101 and HMDB-51, respectively.

1.4 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 details the related background on privacy and efficient machine learning. Chapter 3 proposes a technique for achieving privacy in machine learning by releasing a synthetic dataset instead of a private dataset.

Chapter 4 discusses our novel method for compressing neural networks via mixed-precision quantization. Chapter 5 describes our proposed framework for action recognition on compressed videos. Finally, Chapter 6 concludes this dissertation and points out future research directions that stem from this dissertation.

2. BACKGROUND

This chapter provides some background on Deep Neural Networks (DNN), Differential Privacy (DP), Batch Normalization Layer Statistics, and Neural Network Quantization schemes.

2.1 Neural Networks

A typical DNN consists of interconnected nodes called neurons, which in turn are organized into layers [15]. These layers process input data and transform them into meaningful output. The structure of a neural network usually consists of an input layer, one or more hidden layers, and an output layer [16]. The input layer of a neural network receives data from the user. This data could be anything from numerical values representing pixels in an image to textual information in the form of words or sentences. Each neuron in the input layer corresponds to one feature or input variable. Next, hidden layers are the most significant part of the DNNs [17]. These layers perform complex computations on the input data, transforming it into representations that the network can use to make predictions. Each neuron in a hidden layer is connected to every neuron in the previous layer and has an associated weight and bias. Finally, the output layer of a neural network produces the final result or prediction, based on the processed input data. The number of neurons in the output layer depends on the problem being solved. For example, in a binary classification task [18] (e.g., cat vs. dog), there would be one neuron in the output layer, whereas, in a multi-class classification task [19] (e.g., recognizing handwritten digits [20]), there would be multiple neurons, each corresponding to a different class.

Neural networks learn through a process called training, where they adjust their parameters (weights and biases), based on the input data and the desired output. The algorithm used for training neural networks is back-propagation, which is a form of supervised learning [21]. Back-propagation [22] is an iterative algorithm that allows neural networks to learn and thus improve their performance over time. It works by calculating the gradient of a loss function with respect to the network's parameters, and then updating these parameters in the backward pass of the gradient such that the loss is minimized. Gradient descent [23] is a popular optimization algorithm used for back-propagation in order to update the weights

and biases of a neural network. It works by iteratively adjusting the parameters in the direction that minimizes the loss function.

The weights in neural networks represent the strength of connections between neurons. During the training process, these weights are updated using gradient descent [24] to minimize the difference between the network's predictions and the targets. The weight update rule in neural networks is based on the gradient of the loss function with respect to the weights. This gradient indicates how much the loss should change if we want to make a small adjustment to the weights. By moving the weights in the opposite direction of the gradient, we can minimize the loss and improve the network's performance. The learning rate is a hyperparameter that determines the size of the steps taken during gradient descent [25]. Choosing the right learning rate is important for the convergence of the optimization process. If the learning rate is too small, the training process might be slow, whereas if it is too large, the algorithm may fail to converge [26]. In practice, instead of computing the gradient on the entire training dataset, we can use mini-batch gradient descent [27], which computes the gradient on small batches of data. This approach helps to speed up the training process and makes it more computationally efficient.

Neural networks consist of different types of layers, each used for a specific purpose in the overall computation. Some common types of layers include Dense (Fully Connected) Layers, Convolutional Layers, Recurrent Layers, and Pooling Layers. A dense layer is the simplest type of layer in a neural network, where each neuron is connected to every neuron in the previous layer [28]. This layer is used for general-purpose computation and is often found in the hidden layers of the network. A convolutional layer is commonly used for image processing tasks [29]. It applies convolutional filters to the input data, extracting spatial features such as edges, textures, and shapes. This layer helps the network learn hierarchical representations of the input images. A recurrent layer is used for processing sequential data, such as time series or natural language [30]. It maintains an internal state that is updated at each time step, allowing the network to capture the temporal dependencies in the data. A pooling layer is used in order to reduce the spatial dimensions of the input data, making the network more computationally efficient [31]. Common pooling operations include max pooling and average pooling, which extract the most important features from the input [31],

[32]. Overall, DNNs are powerful computational models that have revolutionized the field of machine learning and artificial intelligence. By mimicking the structure and functioning of the human brain, neural networks are capable of learning from data and making complex predictions or classifications.

2.2 Differential Privacy

This section gives some background on the definition of Differential Privacy (DP). Differential Privacy (DP) was introduced by [8] and it is a framework for analyzing and quantifying the privacy guarantees provided by data-processing algorithms. Differential privacy aims to quantify an individual's privacy when their data is used for the training of a model. Methods that guarantee differential privacy limit the effect of including an individual's data on the outcome of the model, thus protecting the privacy of individuals in the dataset [33].

There are three main principles: privacy protection, quantifiable privacy, and data utility. Privacy protection implies that the primary goal of differential privacy is to protect the sensitive information of individuals, while still allowing useful insights to be extracted from the data. Quantifiable privacy means that it provides a quantitative measure of privacy loss, allowing organizations and individuals to assess the level of privacy protection provided by a particular algorithm or dataset. The data utility refers to the case of preserving differential privacy, while it also aims to maintain the utility of the data for legitimate analysis and decision-making purposes [34].

At the core of differential privacy is the notion of a "privacy budget" or "privacy loss parameter" denoted by ϵ. A randomized algorithm $\mathcal{A}$ is considered to be (ϵ, δ) differentially private, if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{A})$ and for all datasets $x, y \in \text{Domain}(\mathcal{A})$ such that $\|x-y\|_1 \leq 1$:

$$\Pr\left[\mathcal{A}(x) \in \mathcal{S}\right] \leq \mathrm{e}^{\epsilon} \cdot \Pr\left[\mathcal{A}(y) \in \mathcal{S}\right] + \delta$$

In the case of deep learning, $\mathcal{A}$ is the training algorithm, and $\mathcal{S}$ is the subset of all possible model parameters that can be output from the training process [8].

Next, we give a brief overview of the DP mechanisms used in the literature and the challenges and limitations. One of the fundamental mechanisms used to achieve differential privacy is the Laplace mechanism [35]. It adds random noise drawn from a Laplace distribution to the output of a query, thereby ensuring differential privacy and preserving data utility. The exponential mechanism is another important tool regarding differential privacy [36]. It selects outputs from a set of possible outcomes based on their "utility". Then, it adds noise proportional to the sensitivity of the utility function. While offering strong privacy guarantees, achieving differential privacy often comes with a trade-off between privacy and utility [37]. Adding noise to data can affect the accuracy of analysis and introduce errors, especially in cases where precise results are required.

Overall, Differential privacy offers a rigorous framework for protecting individual privacy and allows a meaningful analysis of sensitive data. By quantifying privacy guarantees and providing mechanisms to achieve them, it has become a cornerstone of privacy-preserving data analysis and holds great promise for guaranteeing privacy in a highly data-driven world.

2.3 Batch Normalization Layer Statistics

Batch Normalization (BN) [38] was introduced as a technique to help train DNNs. One challenge with deep neural networks (DNNs) is that their training involves sequentially updating layers, beginning at the output and moving toward the input. This process relies on estimating the error at the output of the DNN, which in turn assumes that the weights in the preceding layers remain constant. DNNs are typically composed of multiple interconnected layers. When we update the parameters of one layer, we usually assume that the parameters of the other layers remain the same. In practice, though, we update all layers at the same time. This implies that all the layers change during an update, making it difficult to get the right target [39].

For example, when we update the weights of the third layer, we expect the weights of the fourth layer's output to have a specific distribution. However, the expected distribution might shift once the weights of the prior layer are modified. This means that the input distribution for each layer changes during the model training as the parameters of earlier

layers are updated. Consequently, this poses challenges for training DNNs with particular types of functions [40].

Furthermore, DNN training is challenging due to their sensitivity to the random weight initialization. A contributing factor to this challenge is that the input distribution to deeper layers in the network may alter after each mini-batch update of the weights. This results in the learning algorithm trying to pursue a constantly shifting target. This phenomenon is called Internal Covariate Shift [38], [41]. BN helps alleviate this challenge by normalizing the inputs for each layer during mini-batch processing. In this case, the DNN converges faster and requires less number of training epochs.

Essentially, BN re-parameterizes the DNN as the DNN's layer's outputs are scaled. Normalization is the process that re-scales the data to have zero mean and one standard deviation. BN ensures that specific units remain normalized, a technique known as "whitening" [42], [43]. Whitening the layer's inputs leads to fixed input distributions, mitigating the internal covariate shift phenomenon. Normalizing the previous layer's activations ensures that the subsequent layer's assumptions about the input's distribution and range remain consistent during weight updates. Moreover, BN can be considered a regularizer, leading to a lower generalization error. Note that BN significantly improves the optimization performance, particularly in CNNs and other DNNS with sigmoidal non-linearities.

Apart from tackling the internal covariate shift phenomenon, evidence suggests that BN's effectiveness may stem from its ability to smooth the optimization function during network training [40]. Specifically, BN fundamentally impacts network training by making the optimization landscape significantly smoother. This results in more predictive gradients, enabling the use of a broader range of learning rates and facilitating DNN convergence [38], [40], [44], [45].

During training, BN is applied by computing each input variable's mean and standard deviation in a layer for every mini-batch. Then, these statistics are used for the normalization. Alternatively, maintaining a running average of the mean and standard deviation across mini-batches is an option. However, it might lead to unstable training. Another approach could be to consider using moving averages for normalization during training. This approach has been found to cause convergence problems. The common practice is to set the

mean and standard deviation for the inputs to each layer using the average values of the entire training dataset at the end of training.

It is worth mentioning that small mini-batch sizes lack a representative distribution of examples from the training dataset. This implies that the normalized inputs can differ significantly between training and inference, resulting in significant performance variations. This issue can be mitigated using Batch Renormalization [46], [47]. Batch Renormalization stabilizes the mean and standard deviation estimates across mini-batches. Batch Renormalization enhances BN by adding a per-dimension correction to ensure consistency between the activations in the training and inference phases. Note that Batch Renormalization can be implemented to the input variables of the first hidden layer or the activations from a hidden layer for deeper layers [48].

BN introduces two additional learnable parameters: a mean (Beta) and a standard deviation (Gamma) [44]. These parameters are used to scale and shift the layers' inputs. Note that the model learns Beta and Gamma as part of the training process. It is worth mentioning that the normalization of each layer input can alter the layer's representational capability. The learned Beta and Gamma parameters help restore this capability. Notably, the back-propagation algorithm is modified to work with the transformed inputs, and the error is utilized to update the new Beta and Gamma parameters learned by the model.

The normalization is performed to the layer's inputs, which can be either the input variables themselves or the output of the activation function from the previous layer [49]. Depending on the chosen activation function, the distribution of inputs to the layer might not follow a Gaussian distribution [50]. Therefore, it might be helpful to first sum and then normalize the activations before they pass through the activation function in the preceding layer [51]. In these cases, the BN transform is added immediately before the non-linearity. Although normalizing the layer's inputs is an option, note that the layer's inputs are typically the output of another non-linearity, which means its distribution shape can change during training. Thus, using the mean and standard deviation for normalization will not contribute to mitigating the internal covariate shift phenomenon.

2.4 Neural Network Quantization

Neural network quantization is a technique used to reduce the memory and computational requirements of DNNs by representing their parameters and activations with lower precision data types. In this section, we will explore the background of neural network quantization, its key components, and the popular methods used in the quantization process. Additionally, we will delve into the uniform quantization technique, its mathematical formulation, and its practical application in optimizing DNNs for deployment on resource-constrained devices.

In the quantization process, a crucial component is the choice of the quantization function. This function maps real values represented in floating-point format to a lower precision range, typically integer values. A popular choice for the quantization function is uniform quantization, which divides the input range into uniformly spaced quantization levels. The uniform quantization function is defined as follows [52], [53]:

$$Q = \left\lfloor \frac{r}{s} \right\rfloor - z \tag{2.1}$$

where Q is the quantization operator, r is the real-valued input, s is the scale factor and z is the zero-point. The $\lfloor\ \rfloor$ operator is the floor operator that maps a real value to an integer through a rounding operation. This method of quantization is known as uniform quantization, as the resulting quantized values (quantization levels) are uniformly spaced. The scaling factor s divides a given range of real values r into a number of partitions and it is defined using the following formula:

$$s = \frac{\beta - \alpha}{2^b - 1} \tag{2.2}$$

where $[\alpha, \beta]$ denotes the clipping range, i.e. a bounded range that we are clipping the real values to, and b is the quantization bit-width. Thus, to determine the scaling factor, the clipping range $[\alpha, \beta]$ should be defined first. For the uniform asymmetric quantization method, the clipping range is not symmetric with respect to the origin. A popular choice for the clipping range is to use the minimum/maximum value of the signal ($\alpha = r_{min}$ and $\beta = r_{max}$). For the uniform symmetric quantization scheme, the clipping range should be

symmetric to the origin and therefore $\alpha = -\beta$. A popular choice is based on the min/max values of the signal: $-\alpha = \beta = max(|r_{max}|, |r_{min}|)$. In practice, per-tensor quantization of weights and activations is applied, meaning that a single set of quantization parameters is used per tensor. This approach simplifies the implementation and reduces the memory overhead associated with storing multiple sets of quantization parameters.

2.5 Video Compression

Video compression aims to reduce the size of video files while preserving visual quality and ensuring efficient transmission and storage [54]. The motivation for video compression comes from the fact that videos consist of vast amounts of data representing frames of images, colors, motion, and audio. Without compression, video files would be prohibitively large, making them cumbersome to store, transmit, and playback. Video compression addresses this challenge by efficiently encoding video data to achieve a trade-off between file size and quality [55].

Video compression removes redundant information from the video while keeping the essential parts intact. There are two main types of compression: lossy [56] and lossless [57]. In lossy compression, some information is permanently discarded to reduce file size. This can result in a slight loss of quality, but there is a significant reduction in file size. Popular lossy compression standards include MPEG [58] and H.264 [59]. In lossless compression, the video is compressed without losing any information. This method is used for applications that require to preservation of every detail, such as for professional video editing or archival purposes [60]. However, lossless compression doesn't shrink file sizes as much as lossy compression.

The common compression techniques are spatial compression, temporal compression, transform coding, and entropy coding. Spatial compression focuses on reducing redundancy within individual frames of the video. This is achieved through spatial prediction and transforms coding [61]. Spatial prediction means that the predicting pixel values are based on neighboring pixels to reduce the amount of information needed to represent each frame. Transform coding converts pixel data into a frequency domain using transforms like the

discrete cosine transform (DCT) [62], which helps concentrate energy in fewer coefficients, facilitating better compression.

Temporal compression exploits similarities between consecutive frames to reduce redundancy across multiple frames. Techniques include Inter-frame Prediction [63] and Motion Compensation [64]. Inter-frame prediction refers to predicting the content of subsequent frames based on previous frames, commonly used in video codecs, such as MPEG [58] and H.264 [59]. Motion Compensation concerns estimating and compensating for motion between frames to encode only the differences, significantly reducing redundancy.

Moreover, transform coding is a fundamental component of video compression, involving transforming the spatial data into a frequency domain. Popular transforms include Discrete Cosine Transform (DCT) [62], [65] and Wavelet Transform [66], [67]. DCT is used in standards such as MPEG and H.264. DCT converts spatial data into frequency coefficients, making it more amenable to compression. Wavelet transform [66] is an alternative to DCT, wavelet transforms are used in newer compression standards like JPEG 2000 [68] and H.265 [69] to achieve higher compression efficiency.

Entropy coding further compresses the transformed video data by assigning shorter codes to more frequent patterns and longer codes to less common patterns [70]. Common entropy coding techniques include Huffman Coding and Arithmetic Coding. Huffman coding [71] assigns variable-length codes to symbols based on their frequency of occurrence, with more frequent symbols represented by shorter codes. Arithmetic coding is similar to Huffman coding but allows for a more efficient representation of probabilities, yielding higher compression ratios [72].

The compressed videos consist of intra-frames (I-frames) and Predictive frames (P-frames). Intra-frames are standalone frames that contain complete image information [73]. They are used as reference points for subsequent frames and provide a starting point for decoding. Intra-frames are encoded independently using spatial compression techniques, such as transform coding, without referencing other frames. Predictive frames, also called P-frames or inter-frames, rely on previous frames for encoding [74]. Instead of storing complete image information, predictive frames only store the differences (motion vectors) between

the current frame and the reference frame. These motion vectors indicate the direction and magnitude of pixel movement between frames.

Motion vectors represent the movement of blocks of pixels between frames [75]. They are important for predicting the motion of objects in a video sequence. Motion estimation algorithms analyze blocks of pixels in reference frames to find the best match for a block in the current frame [76], [77]. The resulting motion vectors encode the direction and distance of pixel movement, enabling efficient prediction of inter-frames. Residuals, also known as prediction errors, represent the differences between the original pixel values and the predicted values based on motion vectors [78]. After motion compensation, residuals capture the remaining image details that cannot be accurately predicted.

Popular video compression standards are MPEG [79], H.264 (AVC) [59], and H.265 (HEVC) [80]. Moving Picture Experts Group (MPEG) standards, including MPEG-2, MPEG-4, and MPEG-H, have been essential in defining compression techniques for various multimedia applications. MPEG-2 revolutionized digital television and DVD video, while MPEG-4 introduced advancements, like object-based coding and scalability. H.264, also known as Advanced Video Coding (AVC), is widely adopted for high-definition video compression, offering efficient compression with excellent visual quality. H.265, or High-Efficiency Video Coding (HEVC), builds upon H.264 to achieve even higher compression efficiency, enabling 4K and 8K video streaming and broadcasting.

Next, we will describe the compression and decompression process [81]. Compression steps:

- Frame Division: The input video stream is divided into keyframes (intra frames) and predictive frames (inter frames).
- Intra-frame Compression: Each intra-frame is encoded independently using spatial compression techniques like transform coding (e.g., DCT). The resulting coefficients are quantized to reduce precision and are entropy encoded for compression.
- Inter-frame Compression: Predictive frames are encoded by estimating motion vectors between the current frame and a reference frame. Motion-compensated prediction is

performed using motion vectors, and residuals are calculated to capture the differences between predicted and actual pixel values.

- Entropy Encoding: The quantized coefficients and residuals are further compressed using entropy coding techniques such as Huffman coding or arithmetic coding to reduce redundancy.

Decompression steps:

- Entropy Decoding: Compressed data is first decoded using the inverse of the entropy coding technique applied during compression, reconstructing quantized coefficients and residuals.
- Inverse Transform: Inverse transforms, such as the inverse discrete cosine transform (IDCT), are applied to decode spatially compressed coefficients and reconstruct pixel values.
- Motion Compensation: Motion vectors are used to predict pixel values in predictive frames, and residuals are added to the predictions to reconstruct the original frame.
- Frame Assembly: Reconstructed frames are assembled in the correct order to reconstruct the original video sequence.

Video compression is a complex yet indispensable process fundamental to modern multimedia applications, facilitating efficient storage, transmission, and playback of video content. Through the use of spatial and temporal compression techniques, along with the transformation and entropy coding methods, video compression standards such as MPEG, H.264, and H.265 have revolutionized our interaction with video across diverse platforms and devices [82], [83]. However, it is worth mentioning that the conventional approach of compressing videos for transmission, followed by decompression for subsequent machine learning tasks such as action recognition, entails a significant time and compute overhead. This prompts the question: Is it feasible to directly perform machine learning tasks on compressed videos? The research detailed in the forthcoming chapters unveils the potential for such an approach.

3. DP-IMGSYN: DATASET ALIGNMENT FOR OBFUSCATED, DIFFERENTIALLY PRIVATE IMAGE SYNTHESIS

Deep Learning has benefited from large statistical data available to the broad community. Large datasets in the domain of image classification [84], object recognition [85], language modeling [86], and recommendation systems [87] have helped these domains make significant advances. However, there are cases in which data cannot be publicly released due to privacy restrictions. To address this issue, several techniques have been proposed in literature which can be categorized into: model release methods [88], and data release methods [89]. Model release methods train a classifier on the private dataset (dataset with privacy restrictions) and release the classifier with privacy guarantees. Data release methods release synthetic data with privacy guarantees instead of private data.

Data release methods offer more flexibility than model release methods [90]–[92]. When releasing data, downstream users have the freedom to choose any model architecture, unlike model release methods. Furthermore, they can combine data from different sources to build better models. Also, in the presence of newly available data, they can retrain the model on new and past data avoiding catastrophic forgetting [93]. However, synthetic images generated using existing data release techniques are visually similar to private images. This poses significant challenges when the image content is visually sensitive and disturbing, for example with certain medical image datasets, content moderation image datasets [94], etc. This leads us to the question: can we achieve the same learning as with private images using *visually dissimilar synthetic images*? For example, can a neural network learn to classify human faces when it is trained with synthetic images depicting places? In our work, we show that such learning is achievable. Formally, we study the problem of *releasing synthetic data for image classification tasks* that satisfy the following properties:

1. The synthetic image must provide $(\epsilon, \delta)-$*Differential Privacy (DP) guarantees.*
2. Retain the *utility* of the private images, i.e. a model trained using synthetic images should result in similar classification accuracy as the model trained on private images.

3. Be *visually dissimilar* to the private images. This is critical to protect the users from viewing visually disturbing and sensitive content.

The visual dissimilarity between synthetic and private images is important for vision data depicting visually disturbing and sensitive content. This requirement is not satisfied by the existing approaches. In particular, existing approaches often use Generative Adversarial Networks [95] with DP guarantees [96]–[99] to generate synthetic images. While GAN-generated synthetic images have DP guarantees and retain the utility of the private images, they are visually similar to private images. Thus, they do not satisfy the third requirement of visual dissimilarity. In addition to this, training a GAN can be challenging due to multiple issues [100] such as vanishing gradients [100], mode collapse [101]–[105], training instability [102], and convergence failure [106].

To address these issues, we propose a non-generative (or discriminative) approach for releasing synthetic images. Our proposed framework, Differentially Private Image Synthesis (DP-ImgSyn) [107], trains a teacher model on the private (sensitive) dataset using a DP training algorithm, such as DP-SGD [88]. Next, a public dataset is selected for distillation [108]. This dataset is used to distill the sensitive dataset. During distillation, we observe that misalignment between the public-private datasets can result in significant performance degradation. To address dataset misalignment, we propose an alignment technique on the public dataset to improve the performance of our framework. Finally, we obtain soft labels using the teacher model and the aligned public dataset. The soft label is the teacher model output (logits) after applying the softmax function. The aligned public dataset (synthetic images) and the corresponding DP-teacher model generated soft labels, are released for training neural networks on image classification tasks. A detailed explanation of the intuition behind our method is provided in Section 3.1.

The synthetic images generated by DP-ImgSyn satisfy (1) privacy: the synthetic images have $(\epsilon, \delta)-$DP guarantees because the teacher model is trained with DP guarantees, (2) utility: this is experimentally verified in Section 3.4. A model trained on the synthetic CelebA-Hair dataset [109], [110] achieves $\approx 99.5\%$ the performance of a model trained with DP on the CelebA-Hair dataset, (3) the synthetic images are visually dissimilar to the private

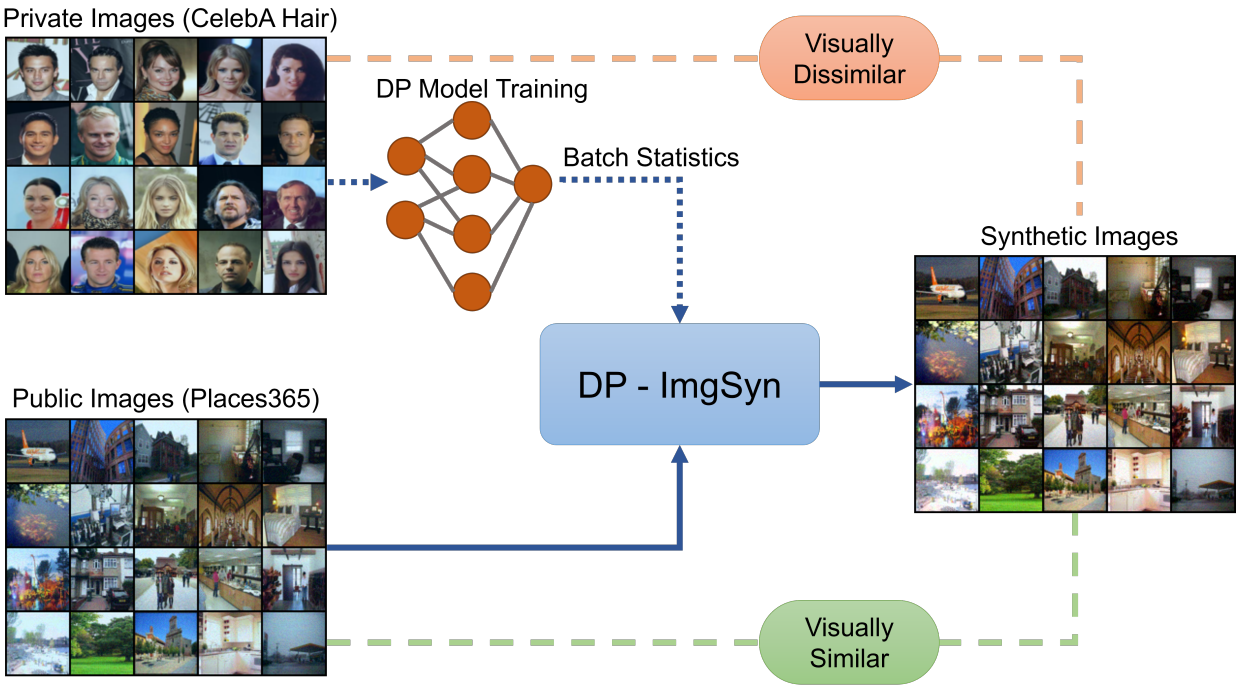


Figure 3.1. Visualization of the private images (CelebA-Hair), public images (Places365), and the DP-ImgSyn generated images (synthetic images). DP-ImgSyn synthesized images are visually dissimilar to private images. DP-ImgSyn is initialized with Places365 (public images/dataset). The public images and the batch statistics of a DP model (trained on the private images) are the inputs to DP-ImgSyn. DP-ImgSyn outputs synthetic images. A model trained on the synthetic images achieves $\approx 99.5\%$ the performance of a model trained with DP on the CelebA-Hair dataset (private images). Figure 3.6 in the Section 3.4.7 contains more visualizations of other datasets.

images. This is observed in Figure 3.1, which visualizes the private dataset (CelebA-Hair), the synthetic images generated by DP-ImgSyn (synthetic images), and the public dataset used for initialization [111]. Further, we experimentally evaluate the synthetic images in terms of classification accuracy on MNIST, FashionMNIST, CelebA-Hair, CelebA-Gender, CIFAR10, and ImageNette datasets. We observe that DP-ImgSyn reduces the public-private distribution misalignment. This alignment improves performance up to $\approx$ **17%** on highly misaligned public-private dataset pairs. Moreover, our method achieves significantly better accuracy (up to $\approx$ **20%**) than state-of-the-art generative methods using a similar DP training algorithm. Note that, the proposed technique is not a new DP mechanism. It is a new

approach for releasing visually dissimilar images, leveraging dataset alignment to obfuscate sensitive data in public datasets.

Overall, our contributions are summarized as follows:

- We propose DP-ImgSyn, a non-generative image synthesis framework that generates a DP-guaranteed dataset for public release. The synthetic images: (1) are DP-guaranteed, (2) have similar utility to the private images, and (3) are visually dissimilar to private images.
- The DP-ImgSyn framework leverages the teacher model's batch normalization layer statistics to address the distribution misalignment between private and public datasets.
- We show the effectiveness of DP-ImgSyn in image classification tasks on various vision datasets. We also show that DP-ImgSyn performs better than state-of-the-art generative methods when using a similar DP training algorithm.

The rest of the chapter is organized as follows: Section 3.1 explains how DP-ImgSyn works and gives the motivation for its use. Section 3.2 gives a brief overview of the related work. Section 3.3 describes the methodology and the steps for DP-ImgSyn. Next, Section 3.4 experimentally evaluates our proposal, and finally, Section 3.5 concludes the chapter.

3.1 DP-ImgSyn: Why and How?

Why? We would like to emphasize that DP-ImgSyn shows that it is possible to learn to classify a dataset that looks very different from the training dataset. Note, that upon synthetic image release, it is up to the user to determine how to use them. Below, we provide examples showing why visual dissimilarity is important in certain classes of applications:

Defense Operational Security (OPSEC) Developing systems for the detection of classified military assets (e.g., a new stealth aircraft) presents significant OPSEC challenges. Typically, only the team directly involved in the asset's development (e.g., the aircraft engineers) possesses the necessary security clearances. To develop a detection system, traditional data-sharing methods would necessitate granting security clearances to additional engineers or declassifying sensitive imagery, both of which introduce security risks [112]. DP-ImgSyn

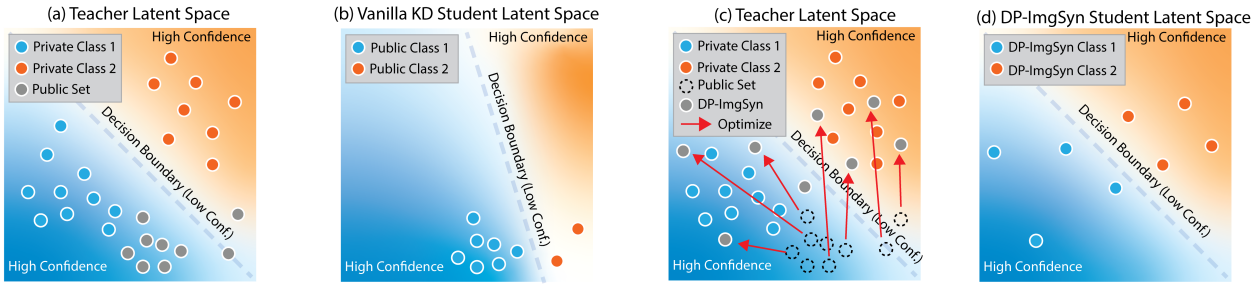


Figure 3.2. (a) Illustrates a cartoon version of the decision boundary of the teacher model and how public images (gray) sample the latent space. (b) Shows the transfer of the decision boundary of the teacher model in (a) to the student model using public images to sample the latent space. The student in (b) learns a decision boundary that is different from the teacher leading to poor knowledge transfer. (c) Shows the effect of optimization/perturbation of the public images with DP-ImgSyn. With DP-ImgSyn the latent space is sampled more effectively. Finally, (d) illustrates the transfer of the decision boundary of the teacher model to the student model, using synthetic images generated by DP-ImgSyn to sample the latent space.

offers a potential mitigation strategy by using synthetic images that are visually obfuscated to protect classified elements while still enabling the development of effective detection models.

Medical datasets are bound by strict privacy regulations (HIPAA [113], PIPEDA [114], and GDPR [115]). While differential privacy (DP) offers robust guarantees, certain datasets may not be suitable for release due to the sensitivity of the body regions depicted [116], [117]. DP-ImgSyn can facilitate the use of such datasets by generating synthetic images that preserve the essential medical information for model training but visually obscuring identifiable details.

Content Moderation DP-ImgSyn potentially addresses privacy and ethical concerns with content moderation by enabling the generation of visually dissimilar synthetic datasets for model training, safeguarding individuals in the original data and mitigating exposure of human moderators to such content [118]. Other applications include automated parental control and movie age rating systems [119].

How? The proposed DP-ImgSyn method aims to learn/transfer the decision boundaries between the teacher and the student model. This is done by sampling the latent space of the teacher model using the public images and having the student model match the decision boundary of the teacher using soft labels. This implies that latent space sampling needs to be effective. The challenge is that public images may not effectively sample the latent space. Thus, our proposed optimization perturbs the public images such that the latent space can be effectively sampled. This leads to better knowledge transfer between the teacher and the student model. Figure 3.2 (a) illustrates a cartoon version of the decision boundary of the teacher model and how a public dataset (gray) samples the latent space. Figure 3.2 (b) shows the transfer of the decision boundary of the teacher model in 3.2 (a) to the student model using the public images to sample the latent space. We see that the student in 3.2 (b) learns a decision boundary different from the teacher leading to poor knowledge transfer. Figure 3.2 (c) shows the effect of optimization of the public images with DP-ImgSyn. With DP-ImgSyn the latent space is sampled more effectively. Finally, Figure 3.2 (d) illustrates the transfer of the decision boundary of the teacher model to the student model, using synthetic images generated by DP-ImgSyn to sample the latent space. The experimental results in Section 3.4 are in line with the DP-ImgSyn intuition. For visualizations on deep neural networks refer to Section 3.4.13.

3.2 Related Work

We consider generating visually dissimilar synthetic images with DP guarantees. While, semi-private learning [120], [121] might seem related to our work, there are fundamental differences. We briefly discuss these differences here. Semi-private learning leverages public data to improve the privacy bounds when learning from private data. Specifically, the private models parameters are updated with public data to improve privacy guarantees and privacy-utility trade-offs. In our work, public data is used for image synthesis and not during the training of the DP neural network. Hence, we do not compare DP-ImgSyn with semi-private learning techniques. Next, we present generative techniques for model and data release.

Regarding generative techniques, they leverage GANs or other generative models by sharing DP-trained models, embeddings, or generating images for releasing datasets while maintaining privacy. Multiple research articles [96]–[98] have proposed training GANs with DP for image synthesis using DP-SGD [88] under various contexts and domains. The authors of GS-WGAN [122] adopt Wasserstein GAN [103] and propose using Wasserstein-1 loss for training. They show that such an approach can distort gradient information more precisely; thus GS-WGANs generate more informative samples. The authors of DataLens [123] leverage GANs to reduce the gradient noise using gradient compression. In a similar direction, to improve information capture from the gradient, the authors of DPGEN [124] deploy an energy-guided network. Note that, DPGEN [124] privacy guarantees are compromised due to conceptual errors, as reported in [125]. They train on synthetic data to indicate the direction of the actual data distribution via the Langevin Markov chain Monte Carlo sampling method. However, since all of these techniques rely on GANs, they are susceptible to training instability of GANs. To address the issues with GAN-based methods, the authors of DP-MERF [126] synthesize images by taking advantage of random feature representations of kernel mean embeddings, while the authors of P3GM [127] use a variant of the private variational autoencoder.

So far, we have discussed generative DP techniques; next, we discuss discriminative DP techniques. Private Aggregation of Teacher Ensembles [89], [128] divides the training data into disjoint sets and assigns them to multiple classifiers (teachers). The teachers are queried with either public or GAN-generated images to obtain the corresponding soft labels [110], [129]. The student network is trained using public or GAN-generated images and their soft labels using knowledge transfer [108]. To maintain the privacy of multiple teacher models, the soft labels of all the teachers are aggregated, and noise is added before they are released for training the student. When PATE employs a GAN for image generation, it encounters the challenges mentioned above related to GANs. When utilizing public images, it struggles to address situations where public and private images are not aligned. Thus, to address the challenges of generative and discriminative data release techniques, we propose a new framework, DP-ImgSyn, for synthetic data release.

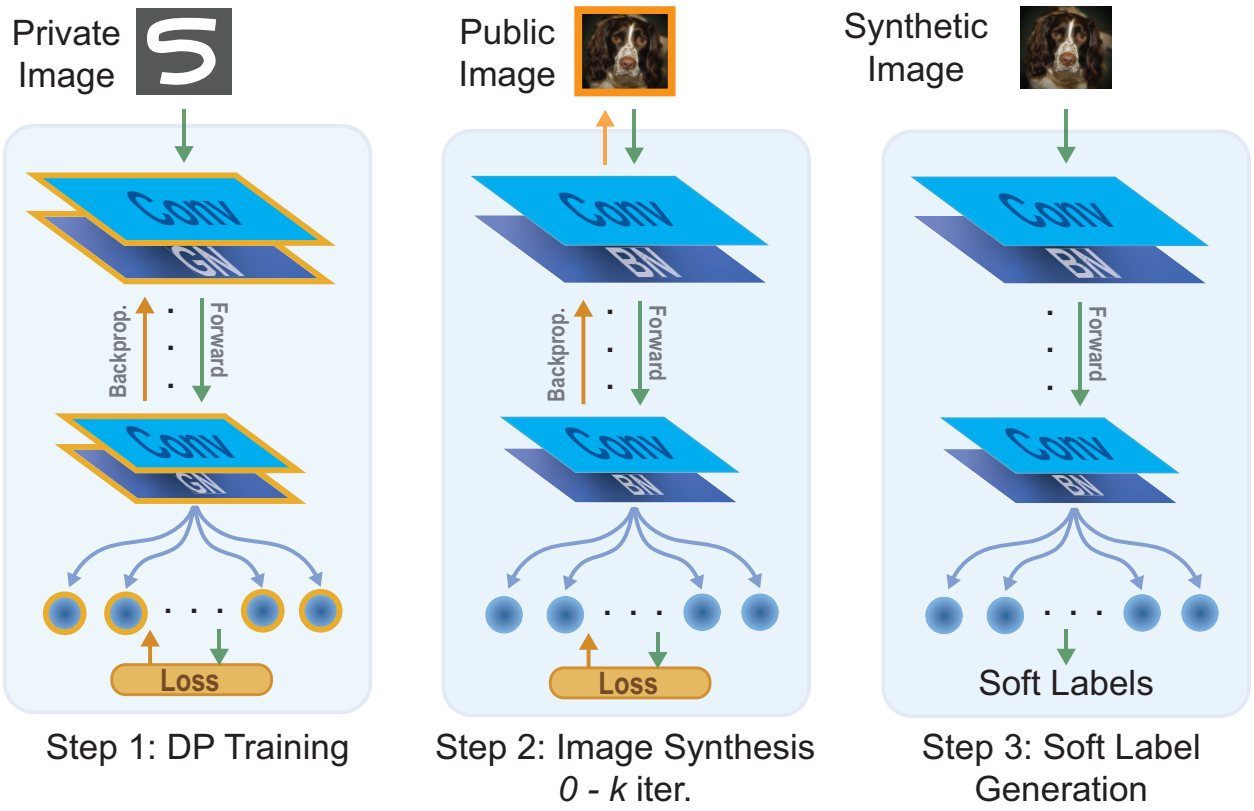


Figure 3.3. Overview of DP-ImgSyn: (1) Train a teacher model using a DP-training scheme. Capture the batch statistics of the model on the private dataset using the proposed DP guaranteed technique. (2) Perform the proposed public dataset alignment to obtain a better-aligned synthetic public dataset. Note that, the public image is optimized (updated), while the parameters of the model stay constant. (3) Generate soft labels. The synthetic images and their soft labels can be publicly released.

3.3 DP-ImgSyn: Differentially Private Image Synthesis

This section introduces our approach's specifics, as Figure 3.3 illustrates. Our method consists of three steps: 1) train a teacher model with a DP training algorithm on the private images, 2) perform optimization on the public dataset to align it to the private dataset and 3) generate soft labels for the aligned synthetic public dataset. Finally, the synthetic images and their corresponding soft labels are publicly released to train a student network.

3.3.1 DP Teacher Model Training

The first step of our method involves training the teacher model utilizing a DP training algorithm. Note that the teacher model is not trained with standard SGD but uses a DP training algorithm to ensure privacy. We select Differentially Private-Stochastic Gradient Descent [88] as the DP training algorithm. DP-SGD trains a neural network with (ϵ, δ)-DP guarantee. Similar to standard SGD, the algorithm converges in multiple training steps. At each training step t, DP-SGD computes the gradient $g_t(x)$ of the loss function with respect to the model parameters for a training image x. Then, it clips each gradient vector g to have a maximum l_2 norm of C. That is, the gradient vector g is replaced by $g/max(1, \|g\|_2/C)$. The clipping ensures that if $\|g\|_2 \leq C$ then g is preserved, whereas if $\|g\|_2 > C$, it gets scaled down to be of norm C. Thus, the contribution of each data point to the batch gradient is bound by a constant C. Noise is added to the gradient $g_t(x) + \mathcal{N}(0, \sigma^2 C^2 I)$ and the descent step $\theta_{t+1} = \theta_t - \eta_t g_t$ is performed, with η_t learning rate. After T iterations, it outputs the $(\epsilon_{train}, \delta)$-DP teacher model.

The next step's synthesis (a.k.a dataset alignment) requires the teacher model's batch statistics. These are obtained from the batch norm layer of the teacher model. However, batch norm layers cannot be used for DP training. This is because the batch norm computes the mean over multiple training data points. Thus, per sample gradient cannot be obtained during training. This implies that the gradient norm cannot be bound, so we cannot provide a DP guarantee. To address this issue, we propose the following DP-guaranteed approach to obtain batch statistics. First, we use group norm layers instead of batch norm layers. Next, we train the teacher model using the previously described DP-SGD. Once trained, we capture the input to all group norm layers, i_{gn}. We clip the input i_{gn} to have a maximum l_2 norm of C and add noise, $\hat{\mathrm{i}}_{gn} = \mathrm{i}_{gn} + \mathcal{N}(0, \sigma^2 C^2 I)$. The noisy input $\hat{\mathrm{i}}_{gn}$ is used to calculate the batch statistics of the private set. This process has the same DP guarantee as the Gaussian Mechanism [130] since it uses the Gaussian DP mechanism described in [130]. Note that obtaining the batch statistics this way consumes some of the privacy budget allocated for training. Thus, when we report our results, the privacy budget ϵ is for the combined training and statistics capture process. To be more specific the reported budget

$\epsilon = composition(\epsilon_{train}, \epsilon_{batch-stats})$. We use the accountant implementation from [131] to calculate the upper bounds on privacy. To ensure correctness, we also verify the upper bounds on privacy using both Opacus [132] and [131]'s implementation.

3.3.2 DP Image Synthesis

Image synthesis (a.k.a dataset alignment) optimizes the public dataset to align with the private dataset to ensure that the synthesized (or aligned) images will have good distillation performance. To perform image synthesis, we collect the layer-wise batch statistics of the private dataset using the DP-guaranteed technique described in Section 3.3.1. The batch statistics consists of the mean $\mu = [\mu_1, \cdots, \mu_L]$ and the variance $\sigma = [\sigma_1, \cdots, \sigma_L]$ from the all the L layers of the teacher model $\mathcal{M}$. Let $\hat{x}$ denote a batch of synthetic images and x_P denote a batch of data sampled from a public dataset D_p. Algorithm 1 summarizes the DP image synthesis process. We initialize $\hat{x}$ with x_P, this corresponds to Line 1 in Algorithm 1. For each data point in $\hat{x}$, we assign a target label y (Line 2 in Algorithm 1). The target labels are uniformly distributed over all the classes, and the assignment is such that we have the same number of images for each class. This step of generating the labels is independent of the private dataset to ensure privacy.

The next step is to perform k iterations of optimization corresponding to Lines 3 - 7. These k iterations optimize $\hat{x}$ to align with the private set. Each iteration consists of a forward pass of $\hat{x}$ through the DP-trained teacher model $\mathcal{M}$ (Line 4 in Algorithm 1). The forward pass is used to obtain the layer-wise batch statistics for $\hat{x}$, $(\mu(\hat{x}), \sigma(\hat{x}))$. The batch statistics and the generated label y are used to calculate the loss described in Equation 3.5 (Line 5 in Algorithm 1). The gradient of the loss $\mathcal{R}$ with respect to $\hat{x}$ ($\nabla_{\hat{x}}\mathcal{R}$) is calculated using back-propagation and is used to update the image $\hat{x}$ (Lines 6 - 7 in Algorithm 1). At the end of k update steps, we have synthesized one batch of aligned images $\hat{x}$. Since the teacher model is DP-trained, and the private dataset batch statistics are obtained with a DP guarantee, image synthesis is also DP-guaranteed.

The loss used to guide the optimization is critical. The total loss $\mathcal{R}_{total}$ consists of the following terms: feature loss $\mathcal{R}_{feature}$, classification loss $\mathcal{R}_{classif}$, total variance loss $\mathcal{R}_{tv}$ and

Algorithm 1: DP Image Synthesis

Input: DP-trained teacher model $\mathcal{M}$; k number of optimization iterations; synthesis learning rate γ_{syn}; batch statistics μ, σ for the private set; batch of public images x_P

Output: $\hat{x}$, one batch of aligned synthetic images

$\hat{x} \leftarrow x_P$
$y \leftarrow$ Target labels for batch $\hat{x}$, uniformly distributed over all the classes
for i $= 1, 2, ..., k$ **do**
    $\mu(\hat{x}), \sigma(\hat{x}) \leftarrow \mathcal{M}(\hat{x})$
    $\mathcal{R} \leftarrow \mathcal{R}_{total}(\hat{x}, y, \mu, \sigma, \mu(\hat{x}), \sigma(\hat{x}))$; `// Compute the loss from Equation 3.5`
    $\nabla_{\hat{x}}\mathcal{R} \leftarrow$ Backward pass
    Update $\hat{x} \leftarrow \hat{x} - \gamma_{syn}\nabla_{\hat{x}}\mathcal{R}$
return $\hat{x}$

l_2 norm loss $\mathcal{R}_{l_2}$. The sum of total variance loss $\mathcal{R}_{tv}$ and the l_2 norm loss $\mathcal{R}_{l_2}$ are referred to as prior loss. Next, we define each of these losses. The feature loss $\mathcal{R}_{feature}$ computes the distance between the batch statistics of the private dataset and the synthetic set $\hat{x}$ and is given by the following equation:

$$\mathcal{R}_{feature}(\mu, \sigma, \mu(\hat{x}), \sigma(\hat{x})) = \sum_{l=1}^{L} \|\mu_l(\hat{x}) - \mu_l\|_2^2 + \|\sigma_l(\hat{x}) - \sigma_l)\|_2^2 \tag{3.1}$$

Where $\hat{x}$ is the synthesized-aligned image, $\mu_l(\hat{x})$ and $\sigma_l(\hat{x})$ are the batch-wise mean and variance estimates of feature maps corresponding to the l^{th} layer when $\hat{x}$ is fed to $\mathcal{M}$, and μ_l and σ_l are the l^{th} layer batch statistics obtained from the private dataset described in Section 3.3.1. The classification loss $\mathcal{R}_{classif}$ is the cross-entropy loss between the teacher output and the target label y and is defined as:

$$\mathcal{R}_{classif}(\hat{x}, y) = \mathcal{L}(p_{\mathcal{M}}(\hat{x}), y) \tag{3.2}$$

where $\mathcal{L}$ is the cross-entropy loss, $p_{\mathcal{M}}(\hat{x})$ is the output of the teacher model $\mathcal{M}$ when $\hat{x}$ is fed as input, and y is the target label. The total variance loss $\mathcal{R}_{tv}$ ensures no sharp

transitions in the synthetic image and restricts the adjacent pixels to have similar values. It is defined as:

$$\mathcal{R}_{tv}(\hat{x}) = \sum_{\mathrm{i,j}}((\hat{x}_{\mathrm{i,j+1}} - \hat{x}_{\mathrm{i,j}})^2 + (\hat{x}_{\mathrm{i+1,j}} - \hat{x}_{\mathrm{i,j}})^2)^{\frac{1}{2}} \tag{3.3}$$

The l_2 norm loss is employed to encourage the image range to remain within a target interval rather than diverging. The l_2 norm loss $\mathcal{R}_{l_2}$ for the $\hat{x}$ is defined as:

$$\mathcal{R}_{l_2}(\hat{x}) = \|\hat{x}\|_2^2 \tag{3.4}$$

The total loss is the sum of the aforementioned losses:

$$\mathcal{R}_{total}(\hat{x}, y, \mu, \sigma) = \alpha_f \mathcal{R}_{feature}(\hat{x}, \mu, \sigma) + \alpha_c \mathcal{R}_{classif}(\hat{x}, y) + \alpha_{tv} \mathcal{R}_{tv}(\hat{x}) + \alpha_{l_2} \mathcal{R}_{l_2}(\hat{x}) \tag{3.5}$$

Each loss term is multiplied by a corresponding scaling factor $\alpha_f, \alpha_c, \alpha_{tv}, \alpha_{l_2}$. The teacher model is not updated during back-propagation, and only $\hat{x}$ is optimized. After k iterations, we obtain the synthetic DP images $\hat{x}$.

3.3.3 DP Image Release

After synthesizing the images, we obtain their corresponding soft labels. The synthetic images are fed to the DP-trained teacher model $\mathcal{M}$, and the corresponding soft labels $\hat{y} = p_{\mathcal{M}}(\hat{x})$ are recorded. Because the teacher model is trained with DP-SGD, querying the teacher to obtain the soft labels does not impose privacy risk. The synthetic images $\hat{x}$, along with their soft labels $\hat{y}$, are publicly released. The student model $\mathcal{S}$ is trained on the synthetic images $\hat{x}$ and their corresponding soft labels $\hat{y}$ using KL-divergence:

$$\min_{\theta} \sum_{x \in \mathcal{X}^s} KL(\hat{y}, p_{\mathcal{S}}(\hat{x})/T) \tag{3.6}$$

KL refers to the Kullback-Leibler divergence, $p_{\mathcal{S}}(\hat{x})$ is the output (soft labels) of the student model when the synthetic image $\hat{x}$ is given as input. T is a scaling temperature value. The synthetic images and their corresponding soft labels are publicly released and can be used to train any neural network. Note that, after synthetic image generation, the public images x_P are no longer needed. Thus, the public images will not be needed during the network training, as the network is trained using synthetic images only.

3.4 Experimental Evaluation

3.4.1 Experimental Setup

To evaluate our proposal, we use the same vision datasets as previous works; specifically, we use MNIST [20], FashionMNIST [133], CIFAR-10 [134], ImageNette [135], CelebA-Hair [109], [110], CelebA-Gender [109], [110], TinyImageNet [136], Places365 [111], LSUN [137], and Textures [138]. Table 3.1 summarizes the statistics for the aforementioned vision datasets. We report the train and test size, the resolution of the images, and the number of classes in each dataset.

We use the following network architectures: ResNet18 [139], ResNet34 [139], VGG11 [140], MobileNetV2 [141], and ShuffleNetV2 [142]. All the models were trained till convergence or privacy budget exhaustion. The detailed hyperparameter settings for image synthesis and model training, computational resources, and dataset statistics are reported in Section 3.4.14. We perform the experiments described in the following sections to evaluate our proposal thoroughly. The models are trained on the synthetic images generated by DP-ImgSyn (training set) and evaluated on the test set of the private dataset (testing set).

3.4.2 Number of Optimization Iterations k

This section studies the effect of the number of optimization iterations on image synthesis and performance.

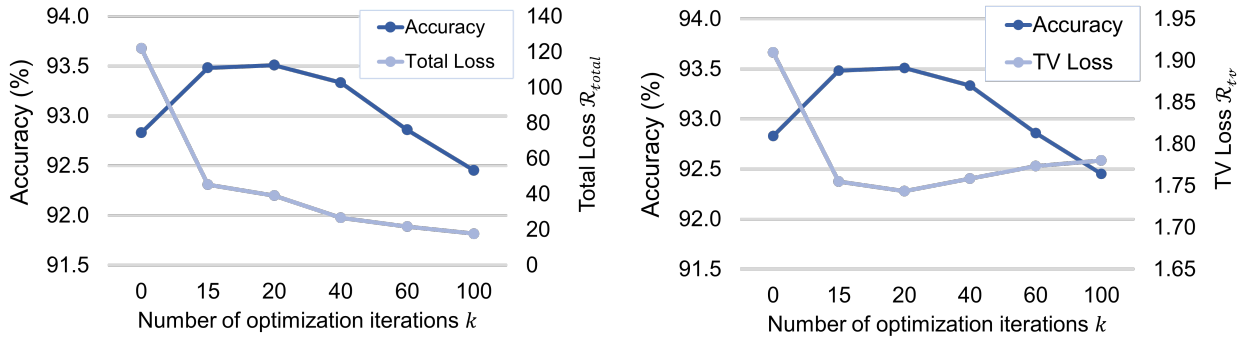
Experiment. We train a ResNet34 till convergence on the private dataset CIFAR-10. Next, we perform the alignment optimization detailed in Section 3.3.2 for various numbers of iterations (i.e., k in DP-ImgSyn Algorithm 1) ranging from 0-100. For the alignment, we

Table 3.1. Dataset statistics, training, and test set sizes for the datasets used.

Dataset	Train Set Size	Test Set Size	Resolution	Number of Classes
MNIST	60,000	10,000	28x28	10
FashionMNIST	60,000	10,000	28x28	10
CIFAR-10	50,000	10,000	32x32	10
Imagenette	10,000	5,000	224x224	10
CelebA-Hair	162,770	19,962	64x64	3
CelebA-Gender	162,770	19,962	64x64	2
TinyImageNet	100,000	10,000	32x32	200
Places365	1,803,460	10,000	32x32	365
LSUN	9,895,373	303,304	64x64	10
Textures	5,640	1,880	224x224	47

use TinyImageNet as the public dataset. The DP-ImgSyn generated dataset is used to train a ResNet18 student model. The student model accuracy and losses versus the number of optimization iterations k are reported. We use privacy budget $\epsilon = \infty$ to isolate all variables. The results are visualized in Figure 3.4.

Results. Figure 3.4a illustrates the accuracy of the student model and the total loss $\mathcal{R}_{total}$ versus the number of optimization iterations k. This is visualized with two y-axes: the left axis for accuracy and the right axis for the total loss $\mathcal{R}_{total}$. Note that the student model accuracy peaks around $k = 20$ iterations. Continuing the optimization by increasing k reduces student model performance. The plot in Figure 3.4b explains the reason for this behavior. Figure 3.4b plots the accuracy of the student model and the total variance loss $\mathcal{R}_{tv}$ versus the number of optimization iterations k. This is visualized with two y-axes, left for accuracy and the right for total variance loss $\mathcal{R}_{tv}$. The loss $\mathcal{R}_{tv}$ expresses the image prior. The image prior ensures no sharp transitions in the synthetic image. It is used in literature [143]–[146] as a proxy for how natural synthesized images are. This plot suggests that the images become more artificial or synthesized as we optimize past a certain threshold. It is observed that around 20 iterations, the $\mathcal{R}_{tv}$ loss starts increasing, and the accuracy starts decreasing.



(a) Accuracy and $\mathcal{R}_{total}$ versus k.

(b) Accuracy and $\mathcal{R}_{tv}$ versus k.

Figure 3.4. Plots for student model accuracy, total loss $\mathcal{R}_{total}$ and total variance loss $\mathcal{R}_{tv}$ versus the number of optimization iterations k. The left axis is for accuracy, and the right is for loss $\mathcal{R}_{total}$ and $\mathcal{R}_{tv}$, respectively. Results suggest early stopping is necessary to optimize accuracy. For this plot, DP-ImgSyn is initialized with the TinyImageNet dataset, and the ResNet34 is the teacher model.

Table 3.2. Accuracy for ResNet18 DP-teacher model with $\epsilon \in \{1, 10\}$ for MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender.

	MNIST	FashionMNIST	CelebA-Hair	CelebA-Gender
$\epsilon = 1$	86.87%	76.27%	79.95%	91.02%
$\epsilon = 10$	96.30%	81.88%	81.74%	92.35%

Conclusion. Minimizing $\mathcal{R}_{total}$ does not guarantee optimal image prior ($\sim \mathcal{R}_{tv}$). However, $\mathcal{R}_{tv}$ significantly impacts student model accuracy. Thus, early stopping is necessary to optimize student model performance.

3.4.3 Privacy and Performance

We evaluate the performance of DP-ImgSyn with 0 iterations and k iterations on various vision datasets as public and private datasets.

Experiment. First, we select a private dataset. We train a teacher ResNet18 model on the private dataset using DP-SGD and capture batch statistics as described in Section 3.3.1. This results in a ResNet18 teacher model with batch statistics having a privacy

budget of ϵ. Next, we select a public dataset and generate the synthetic dataset described in Sections 3.3.2 and 3.3.3. A student ResNet18 model is trained on the synthesized dataset with various public datasets as initialization. We report the accuracy results for privacy budgets $\epsilon \in \{1, 10\}$ on MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender as private datasets. We select public datasets whose size is the same or exceeds the size of the private dataset. This ensures that the synthesized dataset has the same size as the private dataset. When the public dataset is larger than the private set, we randomly sample from the public set to obtain a subset of the same size as the private set. This public subset is used for the synthesis. The number of classes of the public dataset is not required to be the same as the private set, because we use the soft labels of the teacher model. Note that, there is *no* one-to-one correspondence between public and private set images. Section 3.4.10 reports an ablation study of the various loss terms.

Results. Table 3.2 reports the accuracy of the ResNet18 DP-teacher model trained on various datasets. Table 3.3 summarizes the accuracy results for MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender as private datasets and TinyImageNet, Places365, FashionMNIST, MNIST, and LSUN as public datasets. Table 3.3 reports the performance of the student model when performing 0 iterations, reported as DP-ImgSyn(0). The student model is trained on the DP-ImgSyn generated images and the test set is the private test set. We report the performance with early stopping at k_{exp} iterations as DP-ImgSyn(k_{exp}) and the optimal iterations to stop as k_{opt}.

Conclusion. From Table 3.3, we observe that, on average, when the private datasets are aligned over various public datasets, DP-ImgSyn(k) performs similar to DP-ImgSyn(0). For some datasets, we observe $k_{opt} = 0$, i.e., the proposed alignment process does not improve performance much. However, when the datasets are misaligned, like in the case of FashionMNIST and MNIST, we see DP-ImgSyn(k) performs significantly better (**≈17%** improvement in student model accuracy, for FashionMNIST private dataset with MNIST public dataset initialization for $\epsilon \in \{1, 10\}$). Moreover, in Table 3.3 we see that for each private dataset, we use multiple public datasets as initialization and they result in similar accuracy. Thus, the impact of the domain gap (accuracy) between public and private images is minimal. For more details about the interference of the private and public images refer to Section 3.4.11.

Table 3.3. Comparative Table with $\epsilon \in \{1, 10\}$ for MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender as private datasets using TinyImageNet, Places365, FashionMNIST, MNIST, and LSUN as public datasets. Results are mean ± std over three different seeds. The models are trained on the synthetic images generated by DP-ImgSyn (training set) and evaluated on the test set of the private dataset (testing set).

Private Dataset	ϵ	Public Dataset	DP-ImgSyn(0)	DP-ImgSyn(k_{exp})	k_{exp}	k_{opt}
MNIST	$\epsilon = 1$	TinyImageNet	85.83 ± 0.13	**85.98 ± 0.06**	10	10
		Places365	85.00 ± 0.30	**86.01 ± 0.22**	10	10
		FashionMNIST	85.56 ± 0.32	**86.24 ± 0.03**	10	10
	$\epsilon = 10$	TinyImageNet	92.97 ± 0.65	**94.03 ± 0.64**	10	10
		Places365	92.63 ± 0.23	**93.74 ± 0.18**	10	10
		FashionMNIST	93.61 ± 0.37	**93.90 ± 0.30**	10	10
FashionMNIST	$\epsilon = 1$	TinyImageNet	74.99 ± 0.21	74.93 ± 0.02	1	0
		Places365	75.08 ± 0.15	74.94 ± 0.20	1	0
		MNIST	51.58 ± 2.28	**68.38 ± 0.34**	10	10
	$\epsilon = 10$	TinyImageNet	79.04 ± 0.04	78.71 ± 0.20	1	0
		Places365	78.73 ± 0.14	**78.80 ± 0.05**	1	1
		MNIST	54.78 ± 1.15	**71.51 ± 1.49**	10	10
CelebA-Hair	$\epsilon = 1$	LSUN	79.89 ± 0.08	79.42 ± 0.14	1	0
		Places365	79.91 ± 0.08	79.50 ± 0.15	1	0
	$\epsilon = 10$	LSUN	81.31 ± 0.04	79.28 ± 0.20	1	0
		Places365	81.33 ± 0.12	78.73 ± 0.44	1	0
CelebA-Gender	$\epsilon = 1$	LSUN	89.91 ± 0.15	89.17 ± 0.26	1	0
		Places365	90.06 ± 0.05	89.03 ± 0.19	1	0
	$\epsilon = 10$	LSUN	90.99 ± 0.16	89.90 ± 0.26	1	0
		Places365	91.22 ± 0.04	89.27 ± 0.99	1	0

3.4.4 Comparison with other Techniques

Experiment. This section compares our proposal with previous approaches: DP-GAN [96], DP-MERF [126], P3GM [127], DataLens [123] and G-PATE [129]. For a fair comparison, we compare with techniques that use similar DP-training schemes (i.e., variants of DP-SGD). Note, that we exclude comparison with DPGEN [124] because privacy guarantees are compromised due to errors as reported in [125]. The results for other techniques are the best accuracy results reported in prior publications. For our results, we report our best performance results from Table 3.3. Specifically, for $\epsilon = 1$, we use as public dataset

Table 3.4. Comparison Table with state-of-the-art techniques for $\epsilon \in \{1, 10\}$ for MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender. Results for DP-ImgSyn are mean over three different seeds. The best-performing framework is highlighted in bold, and the second-best is underlined. DP-GAN refers to [96], DP-MERF refers to [126], P3GM refers to [127], DataLens refers to [123] and G-PATE refers to [110].

Dataset	ϵ	DP-GAN	DP-MERF	P3GM	DataLens	G-PATE	DP-ImgSyn (ours)
MNIST	1	40.36%	63.67%	<u>73.69%</u>	71.23%	58.80%	**86.24%**
	10	80.11%	67.38%	79.81%	80.88%	<u>80.92%</u>	**94.03%**
FashionMNIST	1	10.53%	58.62%	<u>72.23%</u>	64.78%	58.12%	**75.08%**
	10	60.98%	61.62%	<u>74.80%</u>	70.61%	69.34%	**79.04%**
CelebA-Hair	1	34.47%	44.13%	45.32%	<u>60.61%</u>	49.85%	**79.91%**
	10	39.20%	52.25%	44.89%	<u>62.24%</u>	62.17%	**81.33%**
CelebA-Gender	1	53.30%	59.36%	56.73%	<u>69.96%</u>	67.02%	**90.06%**
	10	52.11%	60.82%	58.84%	<u>72.87%</u>	68.97%	**91.22%**

initialization FashionMNIST, MNIST, Places, and Places for the private datasets MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender, respectively. For $\epsilon = 10$, we use as public dataset initialization TinyImageNet, TinyImageNet, LSUN, and Places for the private datasets MNIST, FashionMNIST, CelebA-Hair, and CelebA-Gender, respectively. Regarding DP-ImgSyn initialization, we could consider the use of GAN-generated images to initialize the DP-ImgSyn. However, doing this would violate the property of visual dissimilarity between the synthetic and the private images, since GAN-generated images are visually similar to the private images. Therefore, we only consider the use of public images and noise (see Section 3.4.10 provides for a more thorough discussion on noise) for the DP-ImgSyn initialization.

Results. Table 3.4 compares the performance of the proposed technique with state-of-the-art techniques.

Conclusion. We observe that our proposed method significantly outperforms generative techniques that use similar DP training schemes (up to $\approx$ **20%**, for both CelebA-Hair and CelebA-Gender for $\epsilon = 1$).

Table 3.5. Comparison Table with state-of-the-art techniques for $\epsilon = 0.2$ for MNIST and FashionMNIST and TinyImageNet as the public dataset initialization. Results for DP-ImgSyn are mean over three different seeds. DP-GAN refers to [96], DP-MERF refers to [126], P3GM refers to [127], DataLens refers to [123] and G-PATE refers to [110].

	DP-GAN	DP-MERF	P3GM	DataLens	G-PATE	DP-ImgSyn (Our)
MNIST	11.04%	62.61%	8.20%	23.44%	22.30%	77.37%
FashionMNIST	10.21%	52.61%	12.80%	22.26%	18.74%	70.63%

Table 3.6. Comparison Table with state-of-the-art techniques using FID score (lower is better) for CelebA dataset with Places365 (A) and LSUN (B) as public datasets for DP-ImgSyn.

Methods	DP-GAN	DP-MERF	P3GM	DataLens	G-PATE	DP-ImgSyn (ours)
ϵ	10^4	10^4	10^4	10	10	10
FID ↓	403.94	327.24	435.60	320.84	305.92	**188.62 (A), 194.88 (B)**

3.4.5 Comparison With SOTA under Strong Privacy Guarantees

In this section, we present experiments with MNIST and FashionMNIST as private datasets with an epsilon value of 0.2 and TinyImageNet as the public dataset initialization. Table 3.5 summarizes the comparison results. Our method demonstrates better accuracy than the best-performing SOTA (DP-MERF) by 14.76% on MNIST and 18.02% on FashionMNIST.

3.4.6 Image Quality

In this section, we use Frechet Inception Distance (FID) [147] to measure image quality which is common practice in literature [123]. FID score is calculated based on the feature representations extracted from an ImageNet pre-trained deep neural network (Inception-v3 model). However, the FID score is not recommended as a metric for grayscale images (MNIST, FashionMNIST) because it involves a network pre-trained on RGB images. This affects the evaluation and the resulting evaluation is not meaningful, as explained in [123].

Figure 3.5. Comparing Private Dataset (left), DP-ImgSyn generated images for $\epsilon = 10$, $\delta = 10^{-5}$ with Places365 as the public dataset initialization (center) and DP-GAN [96] generated images, $\epsilon = 100$, $\delta = 10^{-5}$ (right).

Thus we report the results only on the CelebA dataset. Moreover, we compare private dataset images, DP-ImgSyn generated images, and DP-GAN [96] generated images in Figure 3.5.

Experiment. We evaluate DP-ImgSyn generated images using the FID score and compare with prior works.

Results. Table 3.6 compares the FID score of the generated images from various prior works when using CelebA as the private dataset. When using DP-ImgSyn we initialize it with Places365 (reported as A) and LSUN (reported as B) as the public dataset.

Conclusion. Our proposal achieves a low FID score (lower is better) and outperforms state-of-the-art works.

3.4.7 Visualizations

Figure 3.6 provides illustrations for MNIST, FashionMNIST, and CelebA Gender as private images, the DP-ImgSyn generated images and the corresponding public images and Gaussian Noise that were used as initialization for DP-ImgSyn.

3.4.8 Beyond Generative Methods

This section presents results on higher resolution (224 x 224) and more varied datasets with which generative techniques often have difficulty.

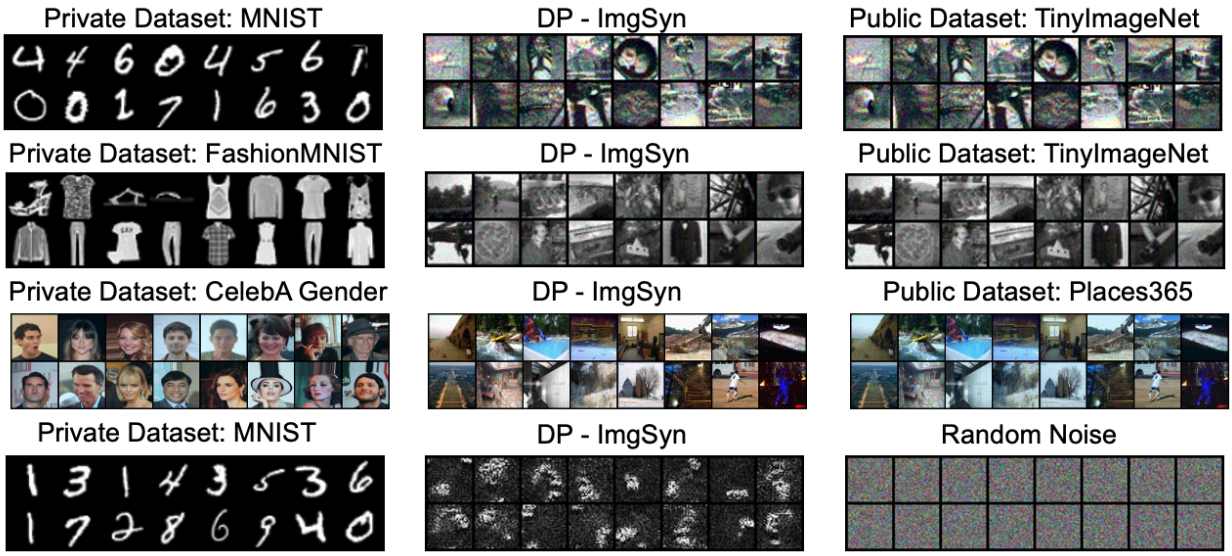


Figure 3.6. We visualize the private, synthetic, and public images (from left to right) for: (first row) MNIST as private dataset with TinyImageNet as public dataset (on TinyImageNet we apply grayscale image transformation because MNIST images are grayscale), (second row) FashionMNIST as private dataset with TinyImageNet as public dataset (on TinyImageNet we apply grayscale image transformation because FashionMNIST images are grayscale), (third row) CelebA Gender as private dataset with Places365 as public dataset, (fourth row) MNIST as private dataset with Gaussian Noise with zero mean and one standard deviation as initialization.

Experiment. We consider ImageNette and CIFAR-10 as private datasets and we use the proposed DP-ImgSyn technique. For ImageNette, we use a resolution of 224 $\times$ 224 with ϵ = 105, initialized with Textures as public dataset; for CIFAR-10, we select $\epsilon = 10$, initialized with TinyImageNet as public dataset. Note that generative methods do not report results on these datasets.

Results. A ResNet18 student model trained on the DP-ImgSyn generated dataset for ImageNette (224 $\times$ 224) achieved **39.38%** accuracy on the test set, while the teacher model achieves 43.26% accuracy. Similarly, for CIFAR-10, a ResNet18 student trained on DP-ImgSyn generated dataset achieved **45.66%** accuracy on the test set, while the teacher model achieves 42.95% accuracy.

Table 3.7. Comparative Table with $\epsilon \in \{1, 10\}$ for MNIST, and CelebA-Gender as private datasets using TinyImageNet, Places365, FashionMNIST, and LSUN and Places365 as public datasets respectively. Results are mean ± standard deviation over three different seeds. The models are trained on the synthetic images generated by DP-ImgSyn (training set) using a ResNet18 as the teacher model. The student model architecture is VGG11, MobileNetV2, ShuffleNetV2, and ResNet18. The student models are evaluated on the test set of the private dataset (testing set).

			Student Model Architecture			
Private Dataset	ϵ	Public Dataset	VGG11	MobileNetV2	ShuffleNetV2	ResNet18
MNIST	$\epsilon = 1$	TinyImageNet	85.56 ± 0.28	84.34 ± 0.70	85.15 ± 0.16	85.98 ± 0.06
		Places365	85.65 ± 0.48	84.02 ± 0.33	85.28 ± 0.22	86.01 ± 0.22
		FashionMNIST	86.37 ± 0.42	84.95 ± 0.28	85.91 ± 0.06	86.24 ± 0.03
	$\epsilon = 10$	TinyImageNet	94.57 ± 0.25	92.08 ± 0.62	94.04 ± 0.09	94.03 ± 0.64
		Places365	93.91 ± 0.26	91.75 ± 0.74	93.22 ± 0.24	93.74 ± 0.18
		FashionMNIST	94.70 ± 0.18	92.66 ± 0.95	92.91 ± 0.91	93.90 ± 0.30
CelebA-Gender	$\epsilon = 1$	LSUN	88.32 ± 0.32	88.31 ± 0.30	89.44 ± 0.17	89.17 ± 0.26
		Places365	88.61 ± 0.38	88.49 ± 0.14	89.16 ± 0.44	89.03 ± 0.19
	$\epsilon = 10$	LSUN	89.36 ± 0.55	88.85 ± 0.25	89.54 ± 0.28	89.90 ± 0.26
		Places365	89.40 ± 0.50	87.85 ± 0.26	89.02 ± 0.02	89.27 ± 0.99

Conclusion. The ImageNette results suggest that DP-ImgSyn is not limited by the image resolution, unlike existing generative techniques. ImageNette and CIFAR-10 results indicate that the DP-ImgSyn technique is better suited to more complex and higher-resolution datasets when compared to generative techniques.

3.4.9 Generalization to Other Network Architectures

This section presents results with various network architectures for the student network to evaluate whether our released dataset can be used to train any network architecture.

Experiment. We use DP-ImgSyn to generate the synthetic images using a ResNet18 as the teacher model. The student model architecture is VGG11 [140], MobileNetV2 [141], ShuffleNetV2 [142], and ResNet18 [139]. For $\epsilon = 1, 10$ we use MNIST and CelebA Gender as the private dataset initialized with TinyImageNet, Places365, FashionMNIST, and LSUN, Places365 respectively.

Results. Table 3.7 summarizes the results across the various student architectures. We observe that the accuracy is similar across the architectures (VGG11, MobileNetV2, ShuffleNetV2), and similar to ResNet18 which has the same architecture as the teacher model.

Conclusion. The outcome of this experiment is that the synthetic images can be used to train any student network architecture. Thus, DP-ImgSyn is not restricted by the teacher model architecture.

3.4.10 Loss Term Ablation Study

In this section, we evaluate the effect of each loss term on the accuracy of the model: $\mathcal{R}_{feature}$, $\mathcal{R}_{classif}$, $\mathcal{R}_{tv}$, and $\mathcal{R}_{l_2}$. The loss term ablation study is summarized in Table 3.8. We present the results for MNIST as the private set with epsilon=10 when using TinyImageNet, Places365, and FashionMNIST as public datasets using three different seeds. To get further insight, we also present results when using Gaussian Noise with mean 0 and standard deviation 1 as initialization for the synthetic images. We make the following observations: 1) from the Gaussian Noise results we see that the feature loss that uses the batch normalization statistics significantly affects the accuracy ($\approx$ 26% accuracy drop when feature loss was excluded in the synthesis loss). 2) when all the loss terms were used this led to a lower standard deviation between the seeds and thus more stability between the runs. The significant effect of the feature loss can be attributed to the value of the scaling coefficient associated with it.

3.4.11 Interference of the features between Public and Private Images

In this section, we quantify the interference of the public image features on the student model test accuracy. To evaluate this, we perform the following experiment: we initialize the synthetic images with Gaussian noise and then perform DP-ImgSyn, using MNIST as the private dataset and epsilon equal to 10. This will set the lower bound on the interference that the public dataset has on the optimization since the starting images are random. We evaluate the accuracy of the student model trained on the synthetic images initialized with

Table 3.8. Ablation study on the loss terms for MNIST as the private dataset, $\epsilon = 10$, and TinyImageNet, Places365, FashionMNIST, and Gaussian Noise as initialization.

$\mathcal{R}_{feature}$	$\mathcal{R}_{classif}$	$\mathcal{R}_{tv}$	$\mathcal{R}_{l_2}$	TinyImageNet	Places365	FashionMNIST	Gaussian Noise
✓	✗	✗	✗	94.30 ± 0.25	93.74 ± 0.13	94.02 ± 0.09	90.07 ± 1.59
✓	✓	✗	✗	94.31 ± 0.09	93.73 ± 0.24	94.19 ± 0.25	89.42 ± 2.36
✓	✓	✓	✗	94.34 ± 0.31	93.74 ± 0.17	93.94 ± 0.28	89.82 ± 1.00
✓	✓	✓	✓	94.03 ± 0.64	93.74 ± 0.18	93.90 ± 0.30	88.53 ± 0.33
✗	✓	✓	✓	93.03 ± 0.63	93.32 ± 0.30	91.70 ± 2.24	63.48 ± 13.62

Table 3.9. The interference of the features of the public images on the private images. The private dataset is MNIST, $\epsilon = 10$. The results for the datasets marked with * are from Table 3.3, and are included in this table as a reference for comparison with the Gaussian Noise.

Private Dataset	ϵ	Initialization	DP-ImgSyn(0)	DP-ImgSyn(k)
MNIST	$\epsilon = 10$	Gaussian Noise	30.12 ± 5.08	**90.55 ± 1.67**
		TinyImageNet*	92.97 ± 0.65	**94.03 ± 0.64**
		Places365*	92.63 ± 0.23	**93.74 ± 0.18**
		FashionMNIST*	93.61 ± 0.37	**93.90 ± 0.30**

Gaussian noise (Table 3.9). We observe that the accuracy drops about 3.4% for the DP-ImgSyn with Gaussian noise initialization compared to best-performing public images. Even though there is interference from the public set, this effect is minimal compared to the effect of DP-ImgSyn. Without DP-ImgSyn, the student model trained using Gaussian noise achieves 30.12% accuracy on MNIST while with DP-ImgSyn it achieves 90.55% (Table 3.9).

3.4.12 Gaussian Noise Initialization with Low-pass Filtering

In this section, we present the results when Gaussian noise is used as initialization with MNIST as the private dataset and $\epsilon = 10$, and then we apply low pass filtering implemented as a Gaussian blur [148]. The results for various kernel sizes are summarized in Table 3.10. Notably, low pass filtering improves the accuracy of the Gaussian noise initialization by up to 1.07% (results for kernel size 3). However, it is still lower ($\approx$ 2.4%) than the best-

Table 3.10. Gaussian Noise with low-pass filtering as initialization with MNIST as the private dataset and $\epsilon = 10$. The low pass filtering is implemented as a Gaussian blur [148] for various kernel sizes.

	Low-pass Filtering			
Without Low Pass Filtering	Kernel Size = 3	Kernel Size = 7	Kernel Size = 11	Kernel Size = 15
90.55 ± 1.67	91.62± 1.35	90.82 ± 0.96	91.10 ± 0.41	91.18 ± 1.23

performing public dataset initialization. Therefore, random noise can as initialization when public images are unavailable, yielding satisfactory performance albeit with some accuracy degradation.

3.4.13 Visualizations of ImgSyn Using a Real Example

In this section, we present a real example of Figure 3.2. In Figure 3.7 we show how ImgSyn can better sample the latent space and improve the decision boundary of the student model. For this experiment, we generated 2D data by sampling from a Uniform distribution in the range $[0, 1]$. The labels were set to 0 or 1 based on the cubic $y = -1.4x^3 + 0.9x1^2 + 0.34$. That is the cubic formed the decision boundary, if $y > 0$ label $= 1$ and vice versa. The cubic was chosen to have a non-linear decision boundary. Since the data is 2D we can visualize the decision boundary in the input space. The top left in Figure 3.7 shows the training data, the two classes shown in green (class 1) and orange (class 2). The teacher model was trained on this classification problem. Please note no DP was used during training since we want to demonstrate ImgSyn. The network confidence is visualized as a heatmap. Blue is high confidence that the sample is in class 1 and yellow is high confidence that the sample is in class 2. The top right shows the same visualization for the student model when applying vanilla knowledge distillation (KD) between the teacher and the student model. It shows the decision boundary at the end of training and the data (illustrated in red) that was used for performing knowledge distillation. For this example, we used two Gaussian clusters centered at $(0.25, 0.25), (0.75, 0.75)$ with a deviation of 0.25 as the dataset to perform knowledge

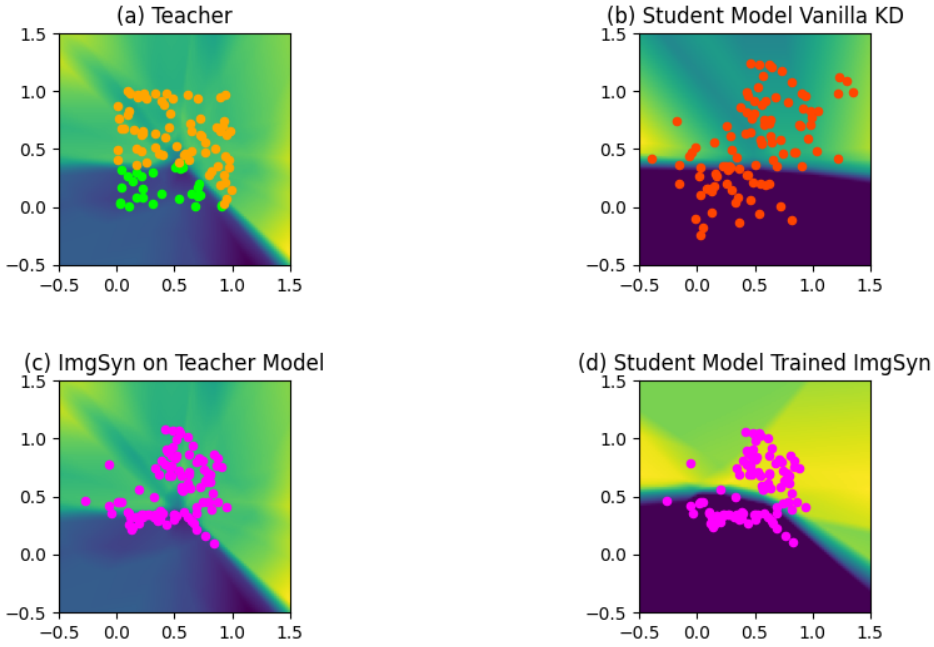


Figure 3.7. Example visualizing deep neural networks decision boundary when using ImgSyn and not using ImgSyn, a data-driven version of Figure 3.2: (a) visualization of the teacher decision boundary, and the training data belonging to two classes. The two classes are shown in green (class 1) and orange (class 2). The teacher network confidence is visualized as a heatmap. Blue illustrates high confidence that the sample is in class 1 and yellow is high confidence that the sample is in class 2, (b) visualization of the decision boundary for the student model when applying vanilla KD between the teacher and the student model, and the Gaussian data used for the vanilla KD (visualized in red) this is analogous to public data, (c) visualization of the teacher decision boundary and the ImgSyn generated data (magenta). The ImgSyn data are generated after applying our ImgSyn on the Gaussian data (illustrated in red color in image (b)) to align it with the source distribution, (d) visualization of the decision boundary of the student model trained on ImgSyn data, and the ImgSyn data (magenta) used for training the student model. The decision boundary of the student model trained on ImgSyn data has a better match to the teacher model than the student trained with on Gaussian data.

distillation from the teacher to the student. Vanilla KD is not able to transfer the decision boundary well from the teacher to the student. The bottom left shows the data after applying

our ImgSyn on the same Gaussian dataset to align it with the source distribution, and the teacher decision boundary when passing the ImgSyn generated data. We see how the points move closer into the range of $[0, 1]$ to match the source distribution and on the bottom right we show the result of training a student model on the ImgSyn data. The decision boundary of the student model trained on ImgSyn data has a better match to the teacher model than the student trained with vanilla KD on Gaussian data.

3.4.14 Training Hyper-parameters

DP-ImgSyn implementation uses the Pytorch [149] framework, and the experiments were conducted on NVIDIA GeForce GTX 1080 Ti with 11 GB of memory with the Ubuntu operating system.

DP Teacher Model Hyper-parameters. For the DP statistics capture for the teacher model (Section 3.3.1), we used the hyper-parameters reported in Table 3.11. The ϵ denotes the privacy budget, η_{tr} is the number of training epochs, Ω_{tr} is the batch size used for DP-SGD training, γ_{tr} is learning rate used for training, C denotes the maximum norm limit for the gradient vector g $(g/max(1, \|g\|_2/C))$, σ controls the amount of noise added to g $(g + \mathcal{N}(0, \sigma^2 C^2 I))$, η_{bn} is the number of epochs used for capturing batch statistics, and Ω_{bn} is the batch size used for capturing batch statistics in Table 3.11. The accuracy results of the corresponding teacher models are reported in Table 3.2.

DP Image Synthesis Hyper-parameters. For the DP Image Synthesis (described in Section 3.3.2, and Algorithm 1), we use the Adam optimizer [150] with synthesis learning rate $\gamma_{syn} = 0.1$, betas $\beta_1 = 0.5$, $\beta_2 = 0.99$. For MNIST and FashionMNIST, we use a batch size of 80; for CelebA-Hair and CelebA-Gender, we use a batch size of 60. Table 3.3 reports the number of optimization iterations for each dataset and privacy budget.

The total loss optimized during image synthesis is the summation of the following losses: $\mathcal{R}_{feature}$, $\mathcal{R}_{classif}$, $\mathcal{R}_{tv}$, and $\mathcal{R}_{l_2}$ (Equation 3.5). The scaling coefficients corresponding to each of these losses are denoted as α_f, α_c, α_{tv}, and α_{l_2}, respectively. Table 3.12 reports the values of the scaling coefficients used in our simulations. Furthermore, we perform an ablation study on the scaling coefficients that control the total loss on MNIST as the private dataset

Table 3.11. Hyper-parameters used for DP training and batch statistics capture experiments on the vision datasets: MNIST, FashionMNIST, CelebA-Hair, CelebA-Gender, CIFAR-10, and ImageNette. The ϵ denotes the privacy budget, η_{tr} is the number of training epochs, Ω_{tr} is the batch size used for DP-SGD training, γ_{tr} is learning rate used for training, C denotes the maximum norm limit for the gradient vector g ($g/max(1, \|g\|_2/C)$), σ controls the amount of noise added to g, η_{bn} is the number of epochs used for capturing batch statistics, and Ω_{bn} is the batch size used for capturing batch statistics

Dataset	ϵ	η_{tr}	Ω_{tr}	γ_{tr}	C	σ	η_{bn}	Ω_{bn}
MNIST	1	4	128	0.01	1.0	0.8	2	64
	10	14	128	0.01	1.0	0.5	2	64
FashionMNIST	1	30	50	0.01	1.2	1	2	64
	10	20	128	0.01	1.2	0.5	2	64
CelebA Hair	1	18	128	0.001	1.0	0.8	5	128
	10	22	128	0.001	1.0	0.45	3	128
CelebA Gender	1	18	128	0.001	1.0	0.8	4	128
	10	22	128	0.001	1.0	0.5	4	128
CIFAR-10	10	12	128	0.001	1.0	0.5	5	128
ImageNette	105	57	8	0.001	1.0	0.3	3	8

for epsilon 10, and TinyImageNet as the public dataset initialization. Specifically, we search the values for each loss scaling factor while keeping the remaining scaling factors constant. Table 3.13 summarizes our results. We observe that our method is robust to the scaling factor hyper-parameter selections. Then, we repeated the experiment using the same setup (MNIST as the private dataset with epsilon set to 10), but with Gaussian noise as initialization. The findings in Table 3.13 consistently affirm our earlier observation, highlighting the robustness of our method to different scaling factor hyper-parameter selections.

Student Model Training on DP-ImgSyn Synthetic Images Hyper-parameters. We use Stochastic Gradient Descent (SGD) optimizer [151] with a learning rate $\eta = 0.1$, momentum 0.9, and weight decay 1e-4 for training a student model on the synthetic images.

Table 3.12. Values of scaling coefficients for the loss terms $\mathcal{R}_{feature}$, $\mathcal{R}_{classif}$, $\mathcal{R}_{tv}$, and $\mathcal{R}_{l_2}$ in Equation 3.5: α_f, α_c, α_{tv}, and α_{l_2}.

α_f	α_c	α_{tv}	α_{l_2}
10	1	2.5e-5	3e-8

Table 3.13. Ablation study on the scaling coefficients α_f, α_c, α_{tv}, and α_{l_2} for MNIST as the private dataset, $\epsilon = 10$, and TinyImageNet and Gaussian Noise as the public dataset initialization.

Loss	Dataset	Hyperparameter Value and Accuracy		
		$\alpha_f = 0.1$	$\alpha_f = 1.0$	$\alpha_f = 10.0$
$R_{feature}$	TinyImageNet	93.99 ± 0.30	94.56 ± 0.15	94.03 ± 0.64
	Gaussian Noise	89.55 ± 0.92	90.30 ± 1.43	88.53 ± 0.33
		$\alpha_c = 0.01$	$\alpha_c = 1.0$	$\alpha_c = 10.0$
$R_{classif}$	TinyImageNet	94.42 ± 0.31	94.03 ± 0.64	94.49 ± 0.04
	Gaussian Noise	90.99 ± 0.86	88.53 ± 0.33	90.81 ± 0.83
		$\alpha_{tv} = 2e\text{-}5$	$\alpha_{tv} = 0.01$	$\alpha_{tv} = 10.0$
R_{tv}	TinyImageNet	94.03 ± 0.64	94.27 ± 0.42	94.32 ± 0.12
	Gaussian Noise	88.53 ± 0.33	89.63 ± 1.90	89.81 ± 2.16
		$\alpha_{l2} = 3e\text{-}8$	$\alpha_{l2} = 0.0001$	$\alpha_{l2} = 0.01$
R_{l2}	TinyImageNet	94.03 ± 0.64	94.45 ± 0.10	94.53 ± 0.16
	Gaussian Noise	88.53 ± 0.33	90.93 ± 1.33	89.87 ± 1.01

We use the multi-step learning rate scheduler with $\gamma = 0.1$ and milestones at 120, 150, and 180 epochs. We train the models for 200 epochs with 256 as batch size. The temperature value from equation 3.6 in our simulations is $T = 100$ for MNIST and FashionMNIST and $T = 10$ for CelebA-Hair and CelebA-Gender.

DP-ImgSyn Label Generation Implementation. We provide the PyTorch code that we use for generating the targets for DP-ImgSyn, given the number of classes and batch size. Since the label generation algorithm is independent of the data, there is no privacy leakage.

```
def generate_labels(num_classes, batch_size):
```

```
    x = torch.arange(num_classes)
    targets = torch.squeeze(x.repeat(1, int(batch_size/num_classes)))
    return targets
```

3.5 Summary

Deep neural networks are state-of-the-art solutions for various tasks in multiple domains, but they require a significant amount of data for training. However, certain data cannot be publicly released due to privacy restrictions. In this chapter, We present a discriminative approach (DP-ImgSyn) for releasing synthetic images that have DP guarantees, maintain the utility of the private images, and are visually dissimilar to the private images. The proposed framework leverages dataset alignment to obfuscate private sensitive images in public images. This alignment/synthesis process improves the performance of even highly misaligned public-private dataset pairs. We observe $\approx$ **17%** improvement in the performance of highly misaligned datasets. Also, we show that the non-generative DP-ImgSyn approach significantly outperforms (up to $\approx$ **20%** improvement in classification accuracy) generative techniques using similar DP-training schemes. Further, we present results on higher resolution (224 x 224) and more varied datasets with which generative techniques often have difficulty. Our findings suggest discriminative (a.k.a non-generative) approaches might better suit synthetic dataset release. However, further research is needed to identify the limits of discriminative and generative techniques.

4. NEURAL NETWORK COMPRESSION VIA MIXED-PRECISION QUANTIZATION

Deep Neural Networks are state-of-the-art solutions for solving a plethora of complex tasks, varying from image classification, object detection, and voice recognition tasks to communication and networking applications [152], [153]. These networks require tremendous computational resources, which might not always be available in resource-constrained devices, to achieve competitive performance on these tasks. A promising solution is the quantization of the weights of the network. This process trains or converts a full-precision network into limited precision while trying to maintain performance.

There exist a great number of works in literature that train a neural network from scratch with limited precision and achieve high compression rates [154]–[156]. However, training in discrete space is challenging, and the convergence is slow [157]. To address this issue, emerging works convert a full precision network into its quantized version [157], [158]. Such techniques are commonly referred to as post-training quantization. Selecting the precision for each layer is a non-trivial problem as the search space is exponential in size. For a network with L layers and m possible bit-width choices for each layer, we have m^L possible choices. Many prior works that avoid the exponential search problem have suggested heuristics to allocate the mixed-precision bit-widths or use a proxy to formulate a solvable optimization problem [159], which adds extra computational overhead. Such methods deploy Bayesian optimization [160], evolutionary search algorithms, [161], [162] calculation of the layer-wise quantization error [159] and adversarial noise computation [158]. We provide more details for the methods mentioned above in section 4.1. Our work belongs in the post-training quantization category that uses different bit precision across the network's layers and reduces the computational overhead by using a multi-layer perceptron (MLP) model for determining the mixed-precision bit widths.

We study the impact of weight quantization on the performance of neural networks. We observe that more aggressive quantization leads to higher divergence between the output of the full precision and the quantized network, and vice versa. Based on this observation, we propose a novel approach to mixed-precision bit-width allocation using a Multi-Layer-

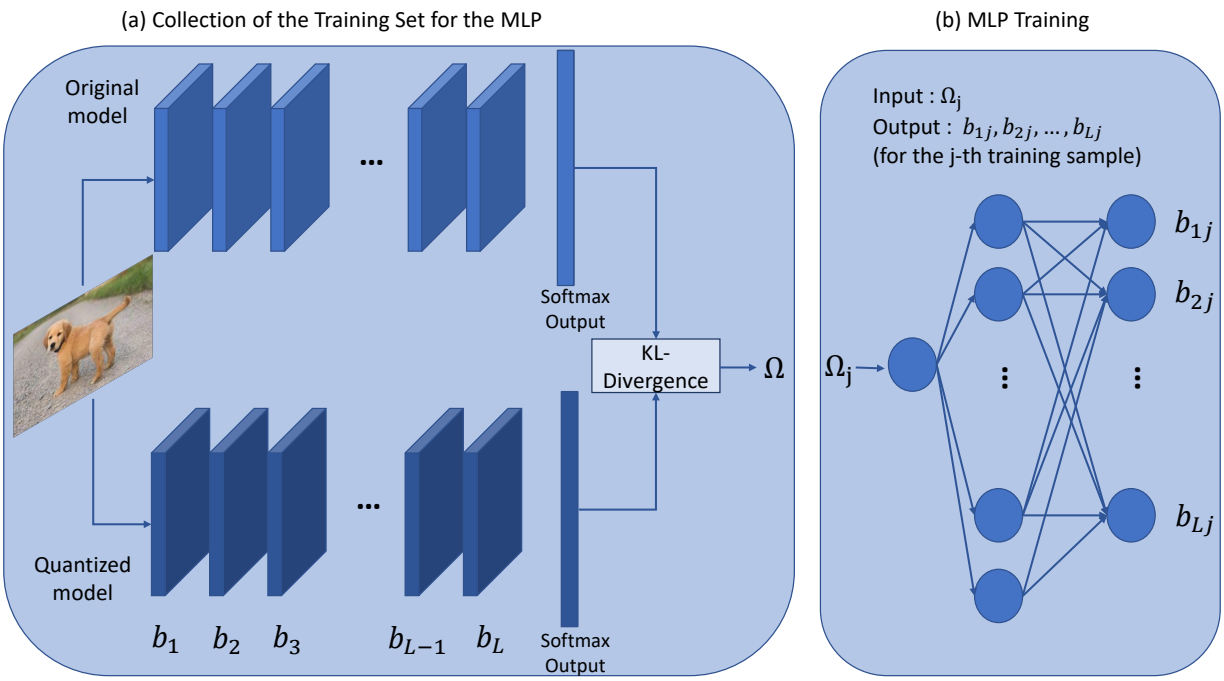


Figure 4.1. Illustration of the proposed framework: (a) the collection of the training set for the MLP. Each layer of the network is quantized with different precision $b_i, i = 1, .., L$, where L is the number of layers in the network. The KL-divergence Ω between the softmax output of the original full precision and the quantized model is computed. This process is repeated S times to collect the entire dataset for the MLP. (b) The MLP is trained using the collected dataset. The input to the MLP is the KL-divergence Ω_j and the output is the bit-width across the layers of the network, for each of the training samples j, j = 1,..., S. The MLP has L output neurons, which is equal to the number of layers of the network intended to be quantized.

Perceptron (MLP) model trained using the Kullback-Leibler (KL) divergence between the softmax output of the full precision and the quantized network [163]. The KL divergence is used as a metric to determine the performance divergence between the two networks. The network intended to be compressed is quantized with multiple mixed-precision bit-width configurations determined using Monte Carlo sampling of the search space. For each sample, the KL divergence at the output is computed. Pairs of the bit-widths (vector) and the KL-divergence are collected, and they constitute the training set for the MLP. The MLP is trained to predict the bit-width configuration (output), given as input the desired

KL-divergence. Since multiple bit-width configurations exist for a given KL-divergence, we introduce a network size penalty to the MLP loss to ensure the network learns to predict configurations leading to the smallest network size. Figure 4.1 illustrates the proposed framework. We evaluate our framework on the ImageNet dataset [84], which is a highly complex dataset for image classification problems, using three different deep neural network architectures, namely VGG16 [140], ResNet50 [139] and GoogLeNet [164]. The experimental results indicate that our method achieves 8x network compression with at most a 0.7% accuracy drop across all the evaluated network architectures.

The rest of this chapter is organized as follows: the related works are summarized in Section 4.1. Section 4.2 presents the impact of quantization on the network and describes the implementation details of the proposed framework: training set collection for the MLP, MLP training, and bit-width allocation across the layers of the network. Section 4.3 reports the experimental evaluation of the proposed method, compares it with state-of-the-art works, and analyzes the computational overhead. Finally, section 4.4 concludes the chapter.

4.1 Related Work

In the domain of network quantization, there exist two main approaches [165]: Quantization - Aware - Training (QAT) and Post-Training Quantization (PTQ). QAT refers to the case of either training a network from scratch with limited precision or fine-tuning for a few epochs with limited precision after network quantization. PTQ refers to the case of quantizing a pre-trained full precision network without retraining it. Note, that methods included in the PTQ category can be data-free or may require a small calibration set, which is readily available [165]. It is worth mentioning that, both QAT and PTQ categories can include methods that allocate mixed-precision weights across the layers of the network. As far as QAT is concerned, there is a plethora of works that trains highly compressed networks [156], [160], [166]–[170]. In [169], the authors model the quantization problem as a discrete constrained optimization problem which is solved using the Alternating Direction Method of Multipliers (ADMM) during training. [168] proposes incremental weight partitioning into two groups, group-wise quantization and retraining. Authors in [167] introduces a quanti-

zation method to reduce this loss by learning a symmetric code book for particular weight subgroups. Furthermore, [166] proposes a quantization scheme that allows inference to be carried out using integer-only arithmetic. In [156] the authors suggest a method that uses low bit-width gradients along with quantized weights and activations during training. Authors in [160] start with a pre-trained full-precision network, deploy Bayesian optimization to allocate the bit precision across the layers of the network, and then fine-tune the network with limited precision for a few epochs. An example of extremely compressed networks are the binary and ternary networks [154], [155], [171]–[173]. However, as mentioned earlier, QAT leads to slow network convergence, and optimization is more difficult in the discrete space [157]. Moreover, the fine-tuning step after quantization can be impractical in many real scenarios, where there is no time to retrain the network after quantization, as the network needs to be deployed immediately [174]. For example, in online learning the network needs to be trained on new data and then deployed immediately.

The second approach (PTQ), including our work, has emerged to address the above challenges. Authors in [157], [175] suggest quantizing the network using equal bit-width across all the layers by using the signal-to-quantization-noise-ratio (SQNR) and the floating and fixed point error probability, respectively. This approach avoids searching the exponential solution space, however, results in sub-optimal bit-width allocation as each layer might have a different impact on the network's performance. Thus, researchers have suggested using mixed-precision quantization across the layers of the network. These works commonly propose using a proxy to formulate a solvable optimization problem for bit-width allocation. In [161], [162], the authors use an evolutionary algorithm-based method to allocate the bit widths. Authors in [159] formulate an optimization problem based on layer-wise quantization errors and they solve it using Lagrangian multipliers. The work in [158] suggests the use of adversarial noise to formulate the optimization problem which is solved by using the Karush-Kuhn-Tucker (KKT) conditions. Further, research has emerged [176] that specializes in the quantization policy for the different hardware architectures. Different from [176], our work does not assume the underlying hardware. Note, that our proposal belongs to the data-free PTQ techniques with mixed-precision allocation across the network's layers. Table 4.1 lists and summarizes the different methods discussed in this section.

Table 4.1. Summary of the related work methods.

	Method's Name	Method's Main Distinctive Characteristic
QAT	TBSQ[169]	Alternating Direction Method of Multipliers
	INQ[168]	Weight Partition and Group-wise Quantization
	SYQ[167]	Learning of a Symmetric Code-book
	IAOI[166]	Integer-only Arithmetic
	DoReFa[156]	Low Bit-width Gradients during training
	CLIP-Q[160]	Bayesian Optimization
	BWN[154]	Binary Network
	BC[155]	Binary Network
	BNN[173]	Binary Network
	TWN[171]	Ternary Network
	FGQ[172]	Ternary Network
PTQ	AGNP[157]	Floating and Fixed Point Error Probability
	FPQ[175]	SQNR
	EMQ[161]	Evolutionary Search Algorithm
	EvoQ[162]	Evolutionary Search Algorithm
	OBA[159]	Lagrange Multipliers
	AQ[158]	KKT conditions
	HAQ[176]	Hardware-based Quantization Policy

Related to network quantization, in the domain of network compression, there are techniques like pruning, that can be applied after or before quantization to compress the network further. These techniques are orthogonal to our proposal and the other post-training techniques and can be implemented on top of them for further network compression. For instance, Han et al. [177] suggest pruning and Huffman coding after quantization. Generally, pruning works [178]–[181] propose the removal of network weights based on some metric (for example the absolute values of the weights, etc.). In this way, the size of the network can be reduced and the network can be more easily deployed in resource-constrained scenarios.

4.2 Multi-Layer Perceptron for Layer-wise Bit-width allocation

4.2.1 Challenges of Quantization

The bit-width allocation across the layers of the network affects its performance and size. When a quantized network is presented with an input it results in errors or deviations from a full precision network. This is a result of errors from quantization accumulating at each layer and reflecting in the output of the network. Different bit-width combinations lead to different impacts on performance and size. Since the search space of all the possible combinations of bit-widths is exponential, finding the optimal solution is non-trivial. The problem is further complicated because two different bit-width combinations may result in similar deviations (many-to-one mapping) in network performance.

Addressing the challenge of quantifying deviations can be done in two ways either by combining layer-wise errors into a single metric or by using a cumulative error metric at the network output. We capture the cumulative impact of the different bit-width combinations on the network's output through our proposed method. Further, the proposed method accounts for the many-to-one problem. If two bit-width combinations have the same impact on the network's performance, our method selects the bit-width with the smallest network size, and it is verified experimentally in section 4.3.2.

4.2.2 Multi-Layer Perceptron

In this section, we present a novel technique to allocate a layer-wise mixed-precision bit-width for quantizing a deep neural network. The proposed method uses a Multi-Layer-Perceptron (MLP) where the input to the MLP is the KL-divergence between the softmax output of the full precision and the quantized network and the output is the bit-width configuration for the compressed network. Our methodology consists of three main steps: sampling the search space to create a custom training set for the MLP, MLP model training, and prediction of the bit-width configuration for the compressed network.

4.2.3 Notation

Let M be a full precision L layered deep neural net, let θ symbolize the parameters of a trained full precision network, and $\tilde{\theta}$ the parameters of a quantized network. The input (image) to the network M is symbolized as x.

4.2.4 Training Set Collection for the MLP

This section describes how the training set for the MLP is collected. Let S be the size of the training set T for the MLP. Each sample in the training set T is a tuple of input and label. The label is a bit-width configuration B_{j} and the input is the corresponding KL-divergence Ω_{j} between the full precision network and quantized network whose bit-width configuration is B_{j}, where j denotes in j^{th} sample in T.

The bit-width configuration $B_{\mathrm{j}} = (b_{\mathrm{ij}})$ is a vector of size L where $\mathrm{i} \in \{1, .., L\}$, $\mathrm{j} \in \{1, .., S\}$ and L is the number of layers of the network M. Each element $b_{\mathrm{ij}} \in B_{\mathrm{j}}$ represents the bit-width for the corresponding layer of the network M. The bit-width configuration B_{j} is sampled from the search space using a Monte Carlo sampling with an additional constraint of $b_{\mathrm{ij}} \in [b_{min}, b_{max}]$. If b_{ij} is extremely low i.e. $b_{\mathrm{ij}} < b_{min}$, this leads to high KL-divergence for most samples in T. Similarly, high b_{ij} i.e. $b_{\mathrm{ij}} > b_{max}$ results in negligible KL-divergence. Therefore, b_{min} and b_{max} are parameters that are determined experimentally for each network. This is done by quantizing all the layers with the same bit-width and evaluating the inference accuracy. The values of b_{min} and b_{max} are chosen such that b_{min} is the maximum bit-width that leads to an accuracy drop of 50% or greater when compared to the full precision network and b_{max} is chosen such that it is the minimum bit-width that maintains the accuracy of the full precision network. Note, that a straightforward selection for b_{max} could be the precision of the original pre-trained model (i.e. $b_{max} = 32$ bits). However, the network can be quantized to 16 bits or sometimes lower bit widths, depending on the model, without accuracy degradation. Thus, we select b_{max} as mentioned previously, to avoid quantizing the network with more bits than needed and at the same time achieving smaller model size. Furthermore, using $b_{max} < 32$ bits reduces the search space during the Monte Carlo sampling. It is worth mentioning that the above process of using the same bit-width across

all the layers happens once to obtain the b_{min} and b_{max}, and eventually each layer of the network M is quantized with different bit-widths (mixed-precision bit-width allocation).

For each sampled bit-width configuration $B_{\mathrm{j}} = (b_{\mathrm{ij}})$ with $\mathrm{j} \in \{1, 2, .., S\}$ and $\mathrm{i} \in \{1, .., L\}$, we quantize the full precision network M with the B_{j} and the KL-divergence Ω_{j} at the output is computed. This process results in a training set T with S samples of the form $T = \{(\Omega_1, B_1), (\Omega_2, B_2), .., (\Omega_S, B_S)\}$, which is used to train the MLP. Similarly, we collect the testing set for the MLP model.

We use the KL divergence between the full precision and the quantized network softmax output as a metric to quantify the divergence between the two networks. The KL-divergence (Ω) is calculated as the average of N images from the training set:

$$\Omega = \frac{1}{N} \sum_{\mathrm{i}=1}^{N} KL(M(\theta, x_{\mathrm{i}}), M(\tilde{\theta}, x_{\mathrm{i}})) \tag{4.1}$$

Note, that we are using the softmax of the model's output to obtain the probability distribution of predicting each class. This probability vector is used for the KL-divergence computation. By definition, the KL-divergence is a measure of how one probability distribution is different from a second reference probability distribution [182].

4.2.5 MLP Training Process

The MLP training is similar to the training of a typical supervised learning task that seeks to minimize the empirical risk:

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{S} \sum_{\mathrm{j}=1}^{S} f(MLP(\theta; \Omega_{\mathrm{j}}), B_{\mathrm{j}}) \tag{4.2}$$

where $f(\cdot, \cdot)$ is the loss function (typically mean squared error or cross-entropy loss) and S is the number of samples in the training set.

However, the standard empirical risk described in Equation 4.2 does not account for the many-to-one mapping problem described in subsection 4.2.1. The many-to-one mapping problem, i.e. the problem when two different bit-width configurations result in the same KL divergence, can be accounted for by ensuring that the MLP learns the configuration that

Algorithm 2: Bit-width Allocation

Input: Full Precision network M with L layers, desired KL-divergence
Output: Bit-width

Sample Collection:
for j $\leftarrow 1$ **to** S **do**
    Generate the random bit-width B_{j}
    Quantize M with B_{j}
    Compute the KL-divergence Ω_{j} for N training images
    Store the pair $(\Omega_{\mathrm{j}}, B_{\mathrm{j}})$
Train MLP with the S samples
Feed the desired KL-divergence to the MLP and obtain the bit-width configuration prediction $B = (b_1, ..., b_L)$ for M

results in the smallest network size. To ensure that the MLP learns the smallest network size, we need to introduce a penalty parameter in Equation 4.2. The modified empirical risk of the MLP includes the size of the quantized network in the loss and it is given by Equation 4.3.

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{S} \sum_{\mathrm{j}=1}^{S} f(MLP(\theta; \Omega_{\mathrm{j}}), B_{\mathrm{j}}) + \frac{\beta}{S} \sum_{\mathrm{j}=1}^{S} \sum_{\mathrm{i}=1}^{L} b_{\mathrm{ij}} * p_{\mathrm{i}} \tag{4.3}$$

where $f(\cdot, \cdot)$ is the loss function (typically mean squared error or cross-entropy loss), S is the number of samples in the training set, β is a scalar and p_{i} is the number of parameters of the ith layer of the network M. The product $b_{\mathrm{ij}} * p_{\mathrm{i}}$ corresponds to the size of the ith layer. The trained MLP predicts the different bit-widths across the layers of the M network (MLP output), given the desired KL divergence (MLP input). The desired KL divergence is user-defined. Algorithm 2 summarizes the steps for the bit-width allocation across the layers of a network M.

4.3 Experimental Results

We present results that deploy our framework to compress the VGG16 [140], the ResNet50 [139], and the GoogLeNet [164] network architectures trained on the ImageNet dataset [84]. The proposed compression framework uses an MLP which consists of two fully connected

layers with 200 neurons in the hidden layer. The number of output neurons is equal to the number of layers of the network to be quantized. For example, for a VGG16 model, which has 16 layers, the MLP will have 16 output neurons. The MLP is trained using empirical risk minimization given by Equation 4.3 and we use mean squared error as the loss function f. For all the models, we use an MLP training set of size $S = 1200$ to train the MLP, by quantizing the model with Monte Carlo sampled bit-widths and computing the KL-divergences between the full precision and the quantized model. For the KL-divergence computation, we observe that $N = 50$ (refer to Equation 4.1) training images are a sufficient number that captures the KL-divergence between the full precision and quantized model. Note, that for the KL-divergence calculation, we do not use the testing images but training images. The activations of the network are set to 8-bit precision for all the simulations. The proposed methodology is implemented using PyTorch [183].

4.3.1 Quantization Scheme

In this work, we use uniform symmetric weight quantization and uniform asymmetric activation quantization for our experiments. We use the ReLU activation function that always results in non-negative values. This causes an imbalance and therefore asymmetric is preferred over symmetric quantization for quantizing the activations. Furthermore, per-tensor quantization of weights and activations is applied, meaning that we use a single set of quantization parameters (quantizer) per tensor.

4.3.2 KL-Divergence

This section studies the relationship between the size of the network and the KL divergence between the full precision and quantized network output. Figure 4.2 (a) is a plot of network size versus KL-divergence and visualizes a subset of the training set T that was used to train the MLP to make the bit-width configuration predictions for the VGG16 [140] network architecture. Each sample in the training set is depicted as a blue dot. The x-axis represents the resulting size of the VGG16 network when quantized with a bit-width configuration $B = (b_1, ..., b_L)$ from the training set T and the y-axis is the corresponding

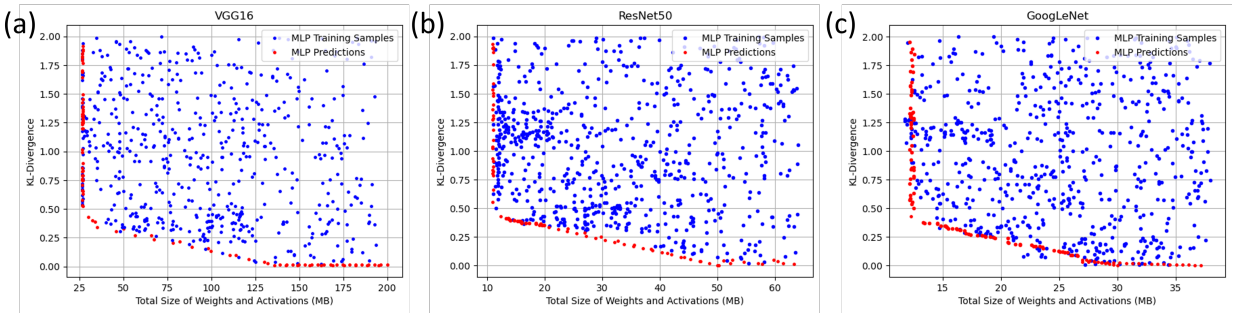


Figure 4.2. The KL-divergence versus the total size of weights and activations (MB) of: (a) VGG16 [140], (b) ResNet50 [139], and (c) GoogLeNet [139] on ImageNet [84]. The x-axis represents the resulting size of the GoogLeNet network when quantized with a bit-width configuration $B = (b_1, ..., b_L)$ from the training set T and the y-axis is the corresponding KL-divergence from T. The blue dots represent the samples from the MLP training set and the red dots represent the MLP's predictions.

KL-divergence from T. We observe that two different bit-width configurations, consequently two different network sizes, may result in the same KL divergence. This is because the order of the bit-widths b_{ij} in the bit-width configuration B plays a significant role, that is some layers of the network are more sensitive to quantization than others, which is leveraged with mixed-precision bit-width allocation.

The proposed training method with the aid of the added penalty described in Equation 4.3 forces the MLP to learn and predict the bit-width configuration that results in the smallest network size. We verify this by plotting the MLP predicted sizes in Figure 4.2 (a). The MLP predictions for various KL-divergence values are marked with red dots. The KL-divergence values are input to the MLP and the predicted bit-width configurations $B = (b_1, ..., b_L)$ are used to compute and plot the resulting network size on the x-axis. We observe from Figure 4.2 (a) that beyond a certain network size, the KL-divergence value saturates near zero. This is expected because in this regime the network is not aggressively quantized and therefore its output does not diverge a lot compared to the original model.

Similarly to Figure 4.2 (a), we plot the training samples T with blue dots and the MLP predictions with red dots for ResNet50, and GoogLeNet architectures in Figures 4.2 (b)

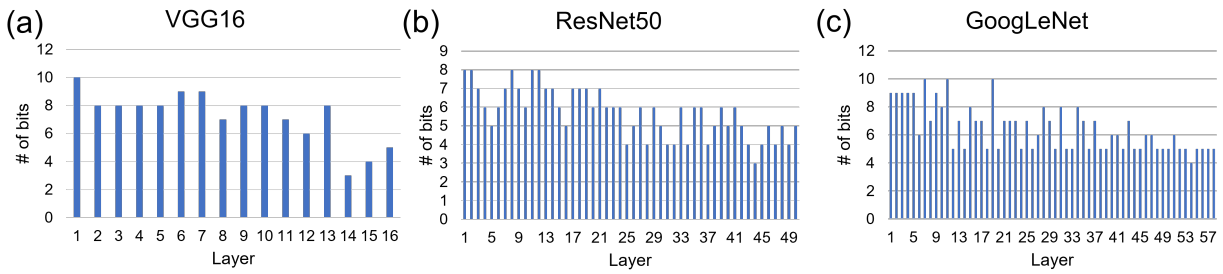


Figure 4.3. The bit-width allocation across the layers of (a) VGG16 [140], (b) ResNet50 [139], and (c) GoogLeNet [139] for KL-divergence = 0.1.

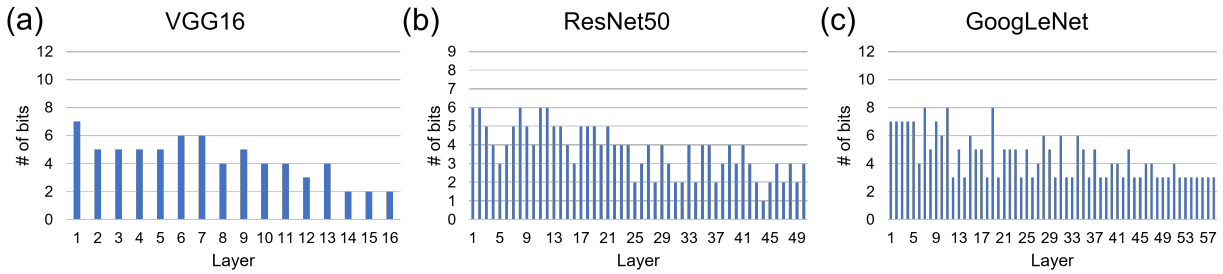


Figure 4.4. The bit-width allocation across the layers of (a) VGG16 [140], (b) ResNet50 [139], and (c) GoogLeNet [139] for KL-divergence = 1.5.

and 4.2 (c), respectively. We make similar observations for ResNet50 and GoogLeNet architectures, i.e. the MLP trained for the ResNet50 and GoogLeNet architectures learns the bit-width configuration leading to the smallest network size.

4.3.3 Bit-width Allocation Analysis

This section presents an analysis of the bit-width allocation across the network's layers obtained using our proposed methodology. Firstly, we select KL-divergence to be equal to 0.1 making the divergence between the full precision and the quantized network negligible. Figure 4.3 (a) illustrates the bit-width allocation for VGG16 that is obtained from the MLP prediction when the input KL-divergence is equal to 0.1. Next, we repeat the above process, but we use KL-divergence equal to 1.5 (Figure 4.4 (a)). In this case, the output of the full precision network significantly diverges from the quantized network. From the bar graphs

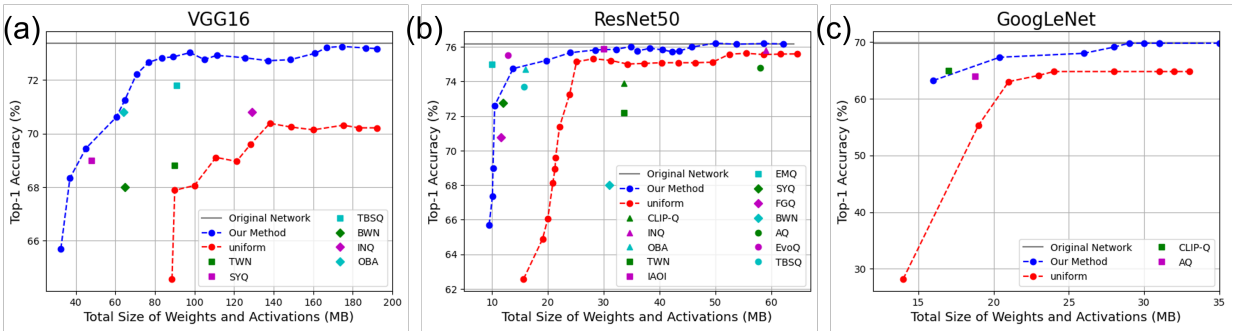


Figure 4.5. The results of state-of-the-art methods on: (a) VGG16 [140], (b) ResNet50 [139], and (c) GoogLeNet [139] over ImageNet [84] dataset.

in Figures 4.3 (a) and 4.4 (a), we observe that the network is more aggressively quantized when KL-divergence is 1.5 compared to KL-divergence of 0.1. This result is in line with our expectation since aggressive quantization introduces more errors and thus the output of the full precision and the quantized model diverges more.

Moreover, we observe that most of the initial layers have higher bit-width than the later ones. Our finding is corroborated by previous work that states that trained networks are more sensitive to their initial layer weights [184]. For ResNet50 and GoogLeNet, we perform a similar analysis as VGG16. The resulting bit-width is depicted in Figures 4.3 (b), 4.4 (b), and 4.3 (c), 4.4 (c), respectively. Note, that similar observations as VGG16, also hold for ResNet50 and GoogLeNet.

4.3.4 Compression Results and SOTA Comparison

In this section we present the compression results that our method achieves for VGG16 [140], ResNet50 [139], and GoogLeNet [164] on ImageNet dataset [84] and we compare our results with state-of-the-art works. Top-1 inference accuracy is the metric used to quantify the performance of the network.

Our method compresses VGG16 [140] up to 6x when compared to the full precision model (32-bit precision weights and activations) with no accuracy drop (see Figure 4.5 (a)). On ResNet50 and GoogLeNet [139], our method achieves up to 4x compression compared

to the full precision model (32-bit precision weights and activations) while maintaining the inference accuracy (see Figures 4.5 (b) and 4.5 (c)).

We compare the results from our experiments with the following state-of-the-art works, including both 1) QAT methods: Binarized Weight Network (BWN) [154], Ternary Weight Network (TWN) [171], Incremental Network Quantization (INQ) [168], Fine-grained Quantization (FGQ) [172], Two-bit Shift Quantization (TBSQ) [169], Integer Arithmetic-only Inference (IAOI) [166], Compression Learning by In-parallel Quantization (CLIP-Q) [160], Symmetric Quantization (SYQ) [167], and 2) PTQ methods: Adaptive Quantization (AQ) [158], Evolutionary quantization of neural networks with mixed-precision (EMQ) [161], Optimizing the Bit Allocation for Network Compression (OBA) [159] and Mixed Precision Quantization of DNNs via Sensitivity Guided Evolutionary Search (EvoQ) [162].

Our work demonstrates better performance than uniform bit-width allocation (visualized with red plot) on VGG16, ResNet50, and GoogLeNet, illustrated in Figures 4.5 (a), 4.5 (b), and 4.5 (c). This is because uniform quantization treats all the layers equally while mixed-precision bit-width allocation captures the impact that each layer has to the network's output more effectively and efficiently. Our method either outperforms or achieves comparable results to state-of-the-art works across different networks architectures, as illustrated in Figures 4.5 (a), 4.5 (b), and 4.5 (c). Our method performs better than all the SOTA works and has performance comparable with SYQ and OBA on VGG16, FGQ, SYQ, IAOI, INQ and OBA on ResNet50, and CLIP-Q on GoogLeNet. EvoQ and EMQ perform better than our method on ResNet50. However, both methods are based on evolutionary search algorithms that evaluate the fitness function multiple times, and also they require a calibration set to perform feature adjustments. This adds extra computational complexity, and we report the exact computational overhead numbers in Section 4.3.5. Note, that our method achieves high compression results with less computational overhead than SOTA works, as it is analyzed in the next section.

4.3.5 Analysis of Computational Overhead

In this section, we will analyze the computational overhead of our method and we will compare it with other quantization methods that require retraining and/or use other forms of optimization. All the experiments were conducted on a system with a Nvidia GTX 1080ti GPU.

The computational overhead is reported in terms of effort factor (ρ). Effort factor is defined as the ratio of the number of FLOPs required to compute the bit-width allocation using a particular method to the number of FLOPs required for one training epoch over the entire training images of the dataset, in our case ImageNet. The effort factor (ρ) is given by Equation 4.4.

$$\rho = \frac{\text{\# of FLOPs of an allocation method}}{\text{\# of FLOPs (forward + backward pass)} * I} \tag{4.4}$$

where I is the number of images of the training set. For the ImageNet dataset, the training set consists of 1.2 million images.

The number of FLOPs required for a training epoch is considered to be three times the number of FLOPs required for a forward-pass [185]. Note, that the number of FLOPs required to compute the bit-width allocation of a method, depends on two parameters: 1) if the method adds a few fine-tuning epochs at the end of bit-width allocation, 2) if the proxy that it used needs additional statistics, for example the layer-wise error over a few training images, to formulate the optimization problem.

The computational overhead of our method arises from the following: 1) the MLP training and, 2) the Monte Carlo sampling and creation of the custom dataset T. For all the experiments, we use a 2-layer MLP with 200 neurons in the hidden layer. As far as MLP training overhead is concerned, we observe that one training epoch requires a forward and backward pass of MLP over the samples of T. VGG16, ResNet50, and GoogLeNet have 16, 50, and 22 layers, respectively, and a training epoch requires a forward and backward pass over the 1.2 million training images of ImageNet. As the MLP size is significantly smaller than the size of VGG16/ResNet50/GoogLeNet and T is substantially smaller than the 1.2

Table 4.2. Comparison of the Computational Overhead of PTQ Methods.

Method	Effort Factor (ρ)
OBA [159]	1x
EMQ [161]	0.3x
EvoQ [162]	0.0022x
This work	0.0016x

million training images of ImageNet; the computational overhead for the MLP training is minimal compared to one training epoch of VGG16/ResNet50/GoogLeNet.

Regarding the computational overhead due to sample collection for the custom dataset T, we observe that this process requires $S*N$ forward passes on VGG16/ResNet50/GoogLeNet, where S is the number of training samples of T and N is the number of training images used for the KL-divergence computation. No backward pass on VGG16/ResNet50/GoogLeNet is required. This computational overhead is significantly smaller than a training epoch for VGG16/ResNet50/GoogLeNet because the number of forward passes ($S*N$) is smaller than the number of forward passes on the entire ImageNet, and no backward pass is required. Therefore, our method has minimal computational overhead ($\rho = 0.0016$x) compared to a training epoch of VGG16/ResNet50/ GoogLeNet on ImageNet.

We compare our computational overhead with other PTQ work [160]–[162], [166], [169]. We do not compare the computational overhead with works belonging to QAT methods [154], [166]–[169], [171], [172], as all the PTQ work, including ours, starts with a pre-trained full precision network. Thus, the QAT methods will have higher computation overhead and it is not a fair comparison. Table 4.2 summarizes the comparison results. We observe that among the PTQ methods, our work adds the lowest computation overhead. In Table 4.2, we have not included the comparison with [158]. This is because it was non-trivial to compute the FLOPs for this work. However, this method requires the computation of dataset-dependent terms whose calculations need 6 hours for the ResNet50 network, as stated in their paper [158]. One training epoch for ResNet50 requires 55 minutes under the same underlying hardware,

which translates to a $\rho = 6.54$x. Thus, [158] is more time-consuming and computationally intensive than one training epoch and therefore more computationally intense compared to our method as well.

4.4 Summary

In this chapter, we proposed a novel simple yet effective bit-width allocation method for network compression that uses an MLP model. We create a custom dataset consisting of pairs of bit-width configurations and KL-divergence to train an MLP model. For a desired KL divergence, which is a proxy for network size, the MLP model predicts the bit-width configuration that the network should be quantized with. The proposed method has little computational overhead compared to other state-of-the-art techniques and achieves up to 6x, 4x, and 4x compression on VGG16, ResNet50, and GoogLeNet respectively with no accuracy degradation compared to the original pretrained full precision network.

Today, there is a trend to move computation from the cloud to the edge. Sensor systems embedded in different devices are an example of such an application [186]–[188]. The edge devices, however, do not have the computational resources that are available on the cloud. To deploy a neural model on the edge it should be compressed to achieve energy efficiency. Our proposal can be used as a low-cost inference method for deploying neural models on edge devices. Moreover, note that our proposal is evaluated on the image classification task. However, this could be applied to any model that performs a supervised task. Evaluating our proposed methodology on other machine learning tasks would be an interesting direction for exploration in the future.

5. PROGRESSIVE KNOWLEDGE DISTILLATION FOR ENHANCED EFFICIENCY AND ACCURACY FOR COMPRESSED VIDEO ACTION RECOGNITION

In typical video classification, the camera captures video data, which is then compressed using video compression algorithms (such as MPEG-4, H.264, and HEVC) [55], [59], [189], transmitted, and subsequently decompressed by the receiver for classification. For such classification, the bottlenecks are the time-consuming compression and decompression steps. To alleviate the above bottleneck, video classification directly from compressed videos using motion vectors (MV), residuals (R), and intra-frames (I-frames) has gained significant attention in recent years [190]–[195]. For example, CoViAR [190] deploys three neural networks, one for processing each compressed video modality (MV, R, I-frames). We observe that networks trained on I-frames converge to a flatter minima, compared to networks trained on R, which in turn are flatter than those trained on MV. Models with flatter minima have been shown to generalize better [196], [197]. We leverage these insights to build a more efficient compressed video classification framework [198].

Firstly, to improve training, we propose progressive knowledge distillation (PKD). The goal of PKD is to progressively distill knowledge from the models with flatter minima to models with less flat minima to improve their performance. Similar to previous works [190], [191], we deploy three neural networks to process the MV, R, and I-frame modalities: MV backbone network, R backbone network, and I-frame backbone network. We propose distilling each modality progressively using more flat networks. That is distill in a sequence using MV, R and I-frame backbones as teachers progressively.

However, this approach presents a cyclic dependency issue. Specifically, training the MV backbone requires all backbones to be trained, and similarly for the R and I-frame backbones. To overcome this problem, we propose using early exit classifiers. Specifically, we train a backbone network using standard cross-entropy (CE) loss for each modality. After the backbone networks have converged, their parameters are frozen. To the frozen backbone, we attach Internal Classifiers (ICs) for early exit, as shown in Figure 5.1. These ICs are trained using PKD. The teacher models are the final classifiers (FCs) of the backbone

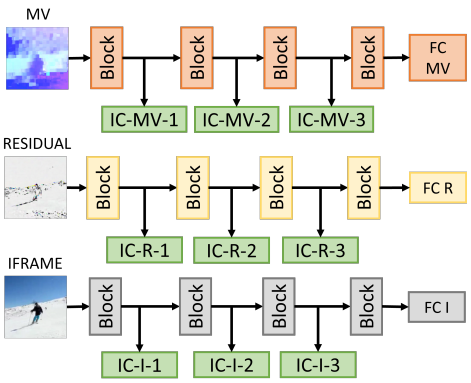


Figure 5.1. Backbone networks with ICs for each video modality.

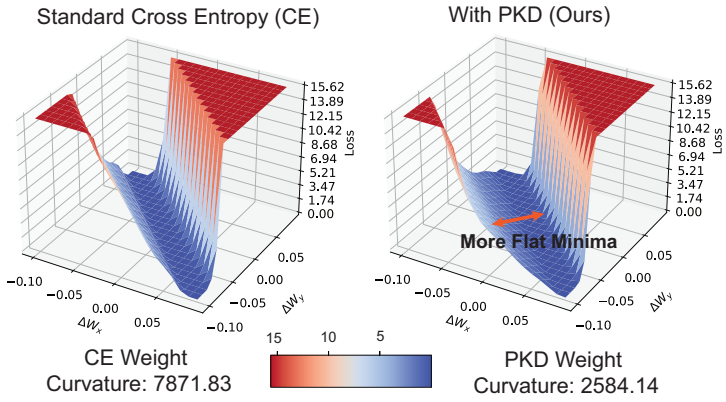


Figure 5.2. Visualizing flatter minima resulting from PKD and the associated loss curvature.

networks of the MV, R, and I-frame. The students are the ICs. Our proposal, PKD trains the ICs in three steps: (1) distill knowledge from the FC of the MV backbone network to all the ICs, (2) distill knowledge from the FC of the R backbone network to all the ICs, and (3) distill knowledge from the FC of the I-frame backbone network to all the ICs. During knowledge distillation, the parameters of the ICs are updated while the parameters of the backbone networks remain frozen. We show that PKD for IC training results in a flatter minima compared to CE (Figure 5.2) leading to better performance and thus, *improved efficiency*.

While PKD tackles training, to improve inference efficiency, we propose Weighted Inference with a Scaled Ensemble (WISE). Consider the l^{th} IC, WISE combines the prediction of the l^{th} IC with predictions from the previous $l-1$ ICs. Each of the l predictions is multiplied by a corresponding scaling factor and aggregated to output a prediction at the l^{th} exit. If the confidence of this prediction is greater than a predefined threshold value, then classification for this video sample terminates at this exit. Otherwise, the classification continues. WISE formulates an optimization problem to determine the scaling factors. Our experimental evaluation demonstrates an increase in classification accuracy by up to 4.28% on UCF-101 and 9.30% on HMDB-51 when using WISE.

In summary, the main contributions are:

- We propose PKD, a novel IC training methodology to transfer knowledge progressively from the FC of the MV, R, and I-frame backbone networks to the ICs, resulting in *improved efficiency*.
- We propose WISE, an efficient inference methodology. WISE leverages cross-modality predictions of previous ICs to improve inference *efficiency*.
- Experimental results of our proposal on UCF-101 and HMDB-51 datasets show up to $\approx 11\%$ IC accuracy improvement with PKD compared to CE. Moreover, we observe up to $\approx 9\%$ accuracy improvement when using WISE.

The rest of the chapter is organized as follows: Section 5.1 gives a brief background on video compression and summarizes compressed video action recognition works. Section 5.2 describes our proposal for efficient training and inference on action recognition tasks using compressed videos. Next, Section 5.3 experimentally evaluates our proposal, and finally, Section 5.4 concludes the chapter.

5.1 Related Work

Video compression algorithms, including established standards like MPEG-4 [79], H.264 [59], and HEVC [80], exploit the recurring similarity between successive frames in a video sequence. These modern codecs typically partition videos into intra-frames (I-frames) and predictive frames (P-frames). I-frames are essentially regular images, preserving spatial information, and are compressed similarly to regular images. P-frames encode temporal information by capturing the "changes" between the frames over time. P-frames consist of MV and R. MVs capture the coarse block-wise movement between frames, while the residuals capture the pixel-wise differences between frames. MV, R, and I-frames are referred as compressed video modalities.

Prior works can be divided into three categories:

1. works that use both compressed and raw (uncompressed) videos during training to enhance accuracy,

2. works that use only compressed videos but have high computational cost, and

3. works that use only compressed videos and focus on reducing the computational cost.

Works that fall in the first category achieve high classification accuracy but come at a significant computational cost. The authors of compressed modality distillation [192] propose using multiple networks and mechanisms (such as ResNet50 networks, Bi-ConvLSTM, self-attention mechanisms, and temporal graphs) to learn and capture the relationship between the raw and the compressed video domains to boost classification accuracy. The authors of MFCD-Net [193] propose a network architecture suitable for processing both the raw and the compressed video. For these works, the benefit of using compressed videos diminishes as access to the raw video during training is required.

CoViAR [190] was one of the earliest compressed video action recognition works in the second category. CoViAR uses the following backbone networks: a ResNet152 to classify the I-frames, a ResNet18 to classify the R, and a ResNet18 to classify the MV. Then, the predictions from these three backbone networks are combined by averaging them. While CoViAR achieves satisfactory classification accuracy, it comes at a significant computational cost. Moreover, TEAM-Net [191] is based on CoViAR, uses similar backbone networks (ResNet50 instead of ResNet152 for the I-frame), and proposes a new module that captures the temporal and spatial relationship of the compressed modalities. CV-C3D [195] extends the 3D convolutional neural networks in the compressed domain. DMC-Net [194] follows an approach similar to CoViAR, but uses an additional network trained using Optical Flow (OF).

Finally, we discuss works that focus on reducing the computational cost. The authors of multi-teacher knowledge distillation [199] use the same backbone networks as CoViAR but replace the ResNet152 with a ResNet18. Their backbone networks are trained using multi-teacher knowledge distillation. However, distilling knowledge from ResNet152 to ResNet18 leads to a noticeable drop in network accuracy. Based on the independent sub-network training [200], the authors of MIMO [201] propose a single network that has multiple-inputs-multiple-outputs (MIMO). The MIMO network concatenates the MV, R, and I-frames and gives them as input to the network to perform a single forward pass. This methodology

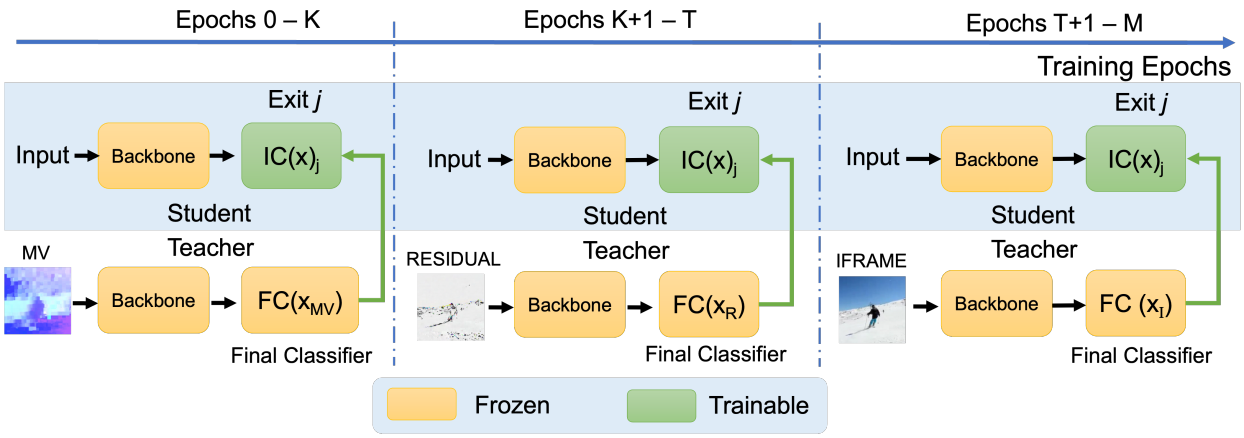


Figure 5.3. PKD IC training Overview: 1) for epoch 0 till K, we perform KD between the IC and the final classifier (FC) of the MV backbone network, 2) for epoch K+1 till T, we perform KD between the IC block and FC of the R backbone network, 3) for epoch T+1 till M, we perform KD between the IC block and the FC of the I-frame backbone network. Note that, the parameters of the backbone networks for the MV, R, and I-frame (illustrated in yellow) are not updated during PKD. The ICs (illustrated in green) are trained independently.

reduces computational cost but imposes accuracy limitations constrained by the network's capacity. Further, the coupled nature of the input imposes the use of an equal number of MV, R, and I-frame as input. This constraint hinders real-time applications, where the entire video stream is not available in advance. Our work falls in this third category and scales between classification accuracy and computational cost.

5.2 Methodology

5.2.1 Progressive Knowledge Distillation (PKD)

PKD is based on the observation that neural networks trained on I-frames converge to a flatter minima compared to models trained on R when converge to flatter minima that models trained on MV. Flatter minima is known to generalize better [196], [197]. To leverage this, we propose PKD which progressively distills knowledge from the compressed video modalities.

The training process consists of two phases. In the first phase, the MV, R, and I-frame backbone networks are trained. The input to the MV backbone network is the motion vectors. Similarly, the R and I-frame backbone networks classify the R and I-frame inputs, respectively. For training the backbone networks, we minimize the CE loss between the predictions and the actual labels. This loss is back-propagated from the FC of the backbone network to the first layer, updating the parameters of the backbone network. Each MV, R and I-frame backbone network is trained independently from the other.

Upon convergence of the backbone network its weight parameters are frozen in preparation for the next phase. In the next phase, the ICs are attached to the trained MV, R, and I-frame backbone networks. The ICs are trained with our proposed PKD methodology. The trained FCs from each backbone network are used as teachers to perform Knowledge Distillation (KD) on the ICs. PKD is based on the idea that transferring knowledge across the compressed video modalities (i.e. MV, R, I-frame) yields more efficient training with flatter minima, compared to using CE or KD from a single classifier. This progressive knowledge transfer consists of three steps, as illustrated in Figure 5.3.

1. MV-KD: KD between the FC of the MV backbone network and the ICs.

2. R-KD: KD between the FC of the R backbone network and the ICs.

3. I-frame-KD: KD between the FC of the I-frame backbone network and the ICs.

Note that during PKD, each IC is trained independently, with each KD loss backpropagating only to the layers of the respective IC. Further, in the description provided we assume that the backbone used is the CoViAR framework, but the same can be applied to other compressed video classification backbones as shown in the experiments section.

5.2.2 Weighted Inference with Scaled Ensemble (WISE)

During inference, we deploy the trained ICs and each IC is a potential early exit. An IC is an exit point if the confidence of the prediction exceeds a certain threshold τ (see Figure 5.4). In this case, the classification process terminates at that exit point; otherwise, it proceeds to the next IC. To that effect, we propose Weighted Inference with Scaled Ensemble (WISE).

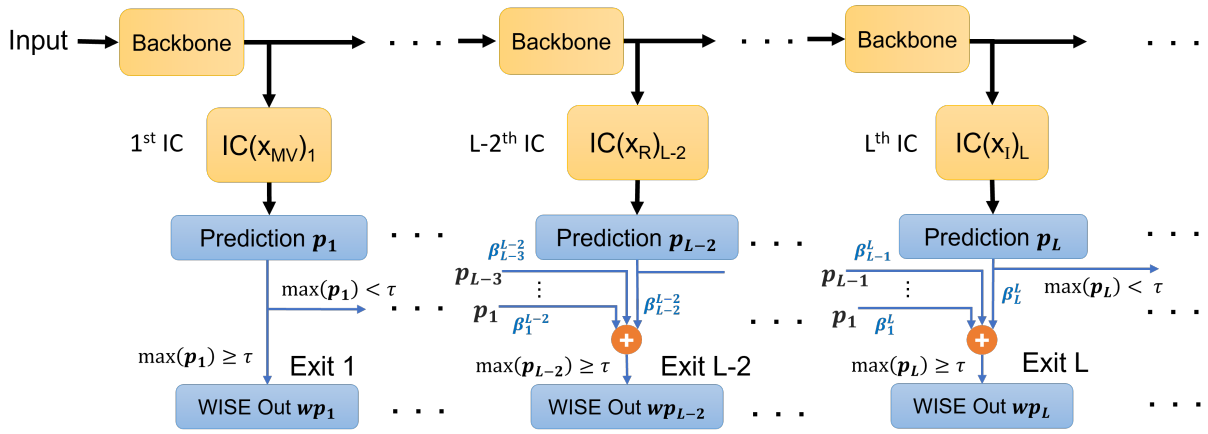


Figure 5.4. WISE Inference Overview: the video sample is evaluated sequentially. The exits might be from different compressed video modality backbones. The previous IC predictions are combined with scaling factors β into an ensemble. If the confidence of the prediction exceeds a certain threshold τ, classification terminates. Otherwise, the next IC is evaluated.

WISE combines the predictions from previous early exits (i.e. ICs where confidence was less than τ). This is achieved by multiplying each IC prediction with scaling factors, followed by aggregation.

For example, let's consider that we are at the L^{th} exit IC. At the L^{th} exit we would have had information from the previous $L-1$ exits. With WISE, we combine the predictions from these ICs and we propose a simple linear combination, where the predictions from the previous ICs are multiplied with some scaling factors β and aggregated to an ensemble prediction. Figure 5.4 illustrates WISE. To find the optimal scaling factors for each IC prediction, we formulate an optimization problem. This optimization problem aims to minimize the CE loss between the ensemble predictions and the actual labels over the training set. The optimization problem is defined as:

$$\min_{\beta^L} \quad -\frac{1}{N}\sum_{\mathrm{i}=1}^{N} t_{\mathrm{i}} \log\left(\sum_{\mathrm{j}=1}^{L} \beta_{\mathrm{j}}^{L} \frac{exp(IC_{\mathrm{i}}^{\mathrm{j}})}{\sum_{m=1}^{m=k} exp(IC_{m}^{\mathrm{j}})}\right) \tag{5.1}$$

where N is the number of video samples in the training set, t_{i} is the actual label for the i^{th} video sample, and L is the L^{th} IC being optimized. Note that $IC^{\mathrm{j}} \in R^k$ is an k dimensional logit from the j^{th} IC for a k class classification problem. IC^{j}_m is the m^{th} element of IC^{j} and $\beta^L = [\beta^L_1, \cdots, \beta^L_L]$ are the scaling factors to combine the IC predictions from the various ICs at the L^{th} exit. Thus, the optimization problem determines the scaling factors β, which combines predictions from various ICs from different modalities. The networks are evaluating sequentially starting from R, then continuing to MV and finally I-frame network. An IC attached to the I-frame backbone has scaling factors β that combine the predictions from previous ICs attached to the MV and R backbone networks as well as the previous ICs of the I-frame backbone network.

5.3 Experimental Evaluation

5.3.1 Datasets and Implementation Details

Datasets. We evaluated our method on two action recognition datasets: UCF-101 [202] and HMDB-51 [203]. UCF-101 contains 13,320 videos from 101 action categories. HMDB-51 contains 6,766 videos from 51 action categories. Each video in both datasets is annotated with one action label. Each dataset has 3 training and testing splits for training and evaluation. We report the average accuracy of the three testing splits over three different seeds for all of our results (i.e. 3 seeds for each split thus, 9 data points in total for one entry).

Architecture and Backbone. We used two types of backbone networks: 1) CoViAR [190], and 2) TEAM-Net [191]. CoViAR consists of a ResNet18 [139] with temporal shift modules (TSMs) [204] for the MV, a ResNet18 with TSM modules for the R, and a ResNet50 [139] with TSMs for the I-frame network. Note that we use a ResNet50 instead of ResNet152 for the CoViAR backbone. The use of TSMs is a standard tool for action recognition tasks and facilitates the adaptation of 2D convolutional networks for video processing without increasing the parameters and floating-point operations (FLOPs). ICs are attached after each ResNet block. TEAM-Net is similar to CoViAR (uses a ResNet18 for MV, a ResNet18 for R, and a ResNet50 for I-frame), but after each block, it inserts an additional block

(TEAM block) that concatenates the features of the three compressed modalities, passes them through a convolutional layer, and then separates them again using three fully connected layers. We attach the ICs after each TEAM block. We focused on the CoViAR backbone for all the experiments, which we found to be more efficient when scaling. However, we also present the results for the TEAM-Net backbone. The datasets and architectures used are standard practice [190], [191], [193], [195]. Details about training and inference are provided in the Subsection 5.3.2.

5.3.2 Training and Inference Hyperparameters

Training. All videos were resized to 240 Œ 320 resolution and compressed to the MPEG4 Part-2 format [58]. We randomly sampled 3 I-frames, 3 MV, and 3 R from videos allocated for the training. The networks for the UCF-101 and HMDB-51 were initialized with pre-trained models on the Kinetics dataset [205]. Further optimization was carried out on UCF-101 and HMDB-51, employing mini-batch training and the Adam optimizer [150] with a weight decay of 0.0001, an epsilon value of 0.001, an initial learning rate of 0.003 for the I-frame input, a learning rate of 0.01 for the MV input, and a learning rate of 0.005 for the R input. The backbone networks were trained for 510 epochs, with a decay in learning rate by a factor of 0.1 at the 150th, 270th, and 390th epochs.

ICs were attached after each residual block of the backbone networks. The IC architecture consists of one convolutional layer and one fully connected layer. The ICs were optimized for 150 epochs using the Adam optimizer, starting with the same initial learning rate as the backbone networks. The learning rate for the ICs underwent decay by a factor of 0.1 at the 50th, 100th, and 150th epochs. The temperature value for the KD was set to 1. The training and evaluation were conducted on a server equipped with an Intel(R) Xeon(R) Silver 4114 CPU and 4 NVIDIA GeForce GTX 1080Ti GPUs with 12 GB of video memory.

Inference. During inference, we use 1 I-frame, 1 MV, and 2 R segments when using the CoViAR backbone, and 8 I-frames, 8 MV, and 8 R segments for the TEAM-Net backbone unless stated otherwise. The frames were cropped into 224 Œ 224 patches and underwent

Table 5.1. Comparing the results of the proposed PKD method for IC1, IC2, and IC3 with Cross Entropy (CE) using MV, R, and I-frames. The reported accuracy corresponds to the average of the three testing splits on UCF-101 and HMDB-51 datasets for results over three different seeds and two backbone frameworks CoViAR and TEAM-Net.

Dataset	Method	Backbones											
		CoViAR									TEAM-Net		
		MV			Residual			I-Frame					
		IC1	IC2	IC3	IC1	IC2	IC3	IC1	IC2	IC3	IC1	IC2	IC3
UCF-101	CE	34.39	47.29	57.43	46.45	60.57	72.63	48.40	57.61	74.11	8.86	28.93	47.41
	Prog KD (Ours)	35.10	48.67	59.20	47.17	62.23	75.34	49.11	60.72	79.98	10.10	31.25	50.20
	Diff. Prog KD – CE	**0.71**	**1.38**	**1.77**	**0.72**	**1.66**	**2.72**	**0.71**	**3.11**	**5.87**	**1.24**	**2.32**	**2.79**
HMDB-51	CE	19.92	26.24	34.58	21.78	31.13	40.19	23.55	27.09	42.42	5.38	18.12	24.40
	Prog KD (Ours)	20.10	27.43	38.08	24.77	35.27	48.07	27.32	35.15	53.84	5.71	19.21	26.04
	Diff. Prog KD – CE	**0.18**	**1.20**	**3.50**	**2.99**	**4.14**	**7.87**	**3.77**	**8.06**	**11.42**	**0.33**	**1.08**	**1.65**

horizontal flipping with 50% probability (i.e., we used only 1 crop). The entire framework was implemented using PyTorch [183].

5.3.3 Evaluating PKD

Experiment. We train the ICs attached to the MV, R, and I-frame backbone networks using two approaches: 1) PKD, and 2) CE. We evaluate the accuracy of each IC on the test set. We use 3 frames for each compressed video modality (I-frame, R, MV) for CoViAR and 8 frames for TEAM-Net. We present the accuracy of the ICs on UCF-101 and HMDB-51 in Table 5.1. Since the backbone networks have 4 blocks, attaching an IC after each block results in 3 ICs per modality (i.e. 3 for each MV, R, and I-frame). Note that the 4^{th} IC is the final classifier and is not trained using PKD or CE, but is trained in the pre-training phase and thus is excluded from this comparison.

Results. The classification accuracy of the ICs are presented in Table 5.1. IC1, IC2, and IC3 correspond to the ICs attached after the first block, second block, and third block respectively for classifying all the video samples. To highlight the improvement of the proposed PKD over CE we also present a row with the difference between CE and PKD. The results in Table 5.1 demonstrate the superior accuracy of ICs trained with PKD using CoViAR as the backbone framework, showing an increase in accuracy of up to 1.77%, 2.72%, and 5.87%,

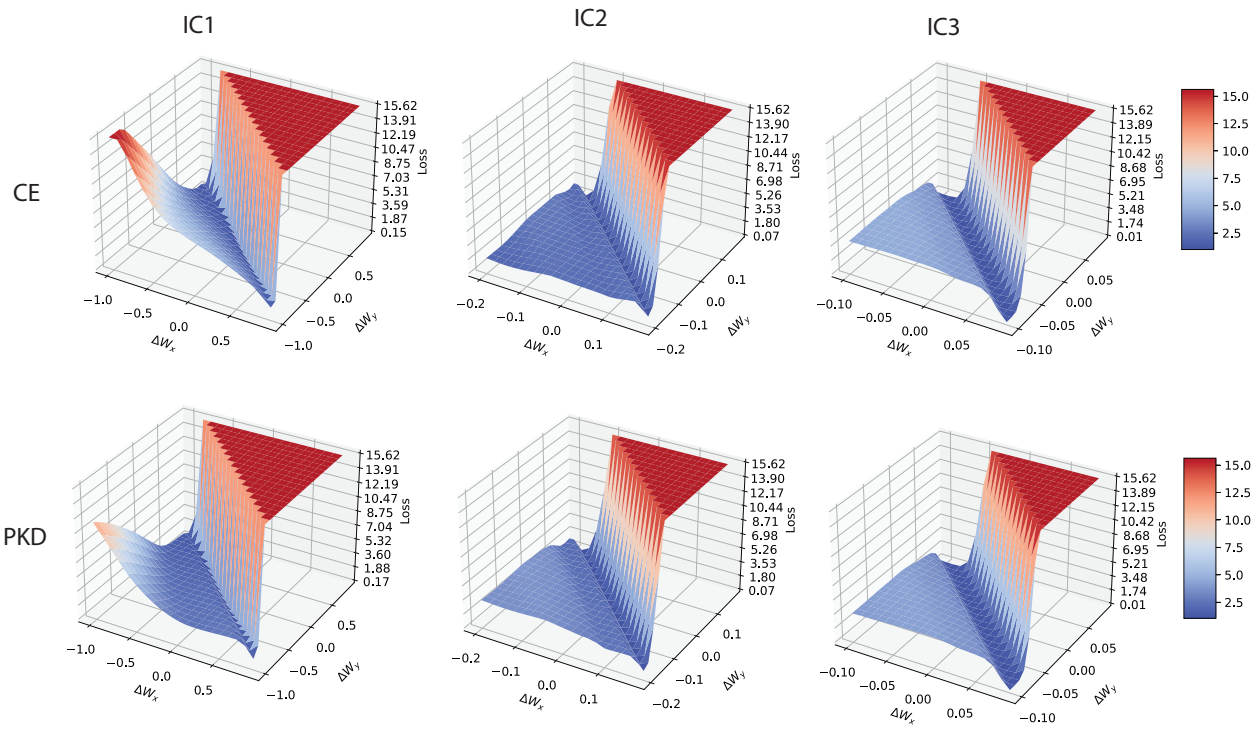


Figure 5.5. Visualizing the loss surface around the minimum of early exit classifiers attached to the MV backbone network when trained using CE vs PKD.

for MV, R, and I-frame, respectively on UCF-101 and 3.50%, 7.87% and 11.42% on HMDB-51 when compared to the ICs trained with CE. When using TEAM-Net as the backbone framework, we observe an increase in IC accuracy of up to 1.24%, 2.32%, and 2.79%, for MV, R, and I-frame, respectively on UCF-101 and 0.33%, 1.08% and 1.65% on HMDB-51, when using PKD compared to training the ICs with CE loss. Note, that this improvement sources from the fact that the ICs trained with PKD result in flatter minima compared to the ICs trained with CE, and thus generalize better. Figure 5.2 visualizes the landscape for IC2 of I-frame. Figures 5.5, 5.6, 5.7 visualize the loss landscape for the ICs of the Motion Vector, Residual, and I-frame network, respectively, trained with CE and PKD.

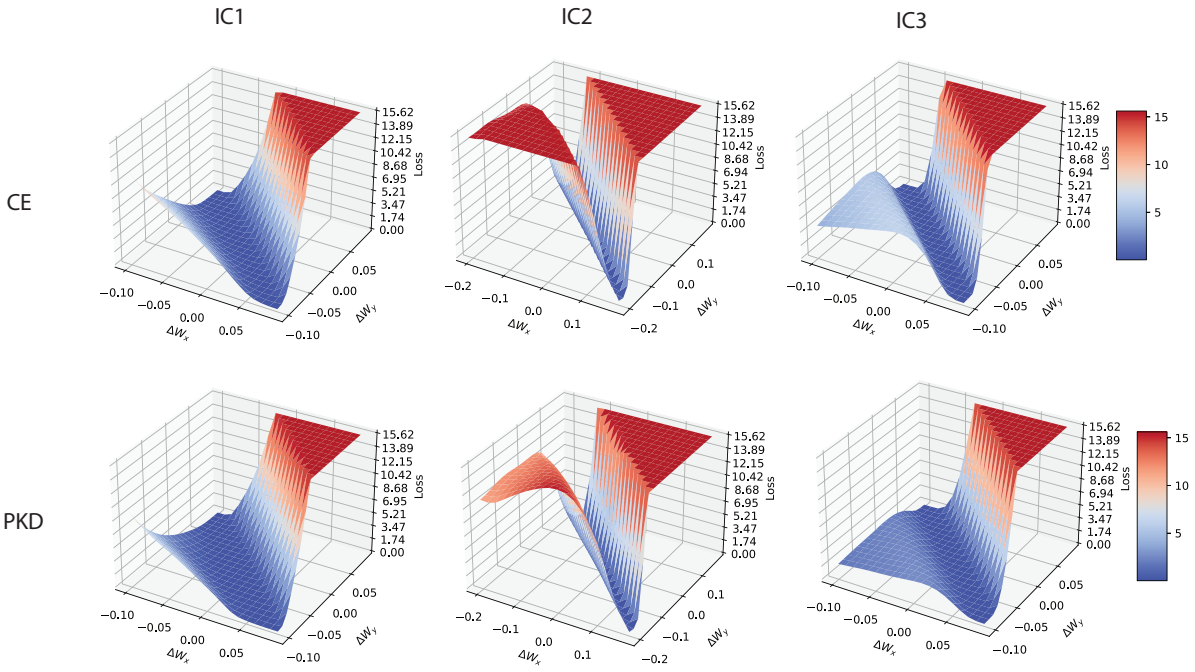


Figure 5.6. Visualizing the loss surface around the minimum of early exit classifiers attached to the Residual backbone network when trained using CE vs PKD.

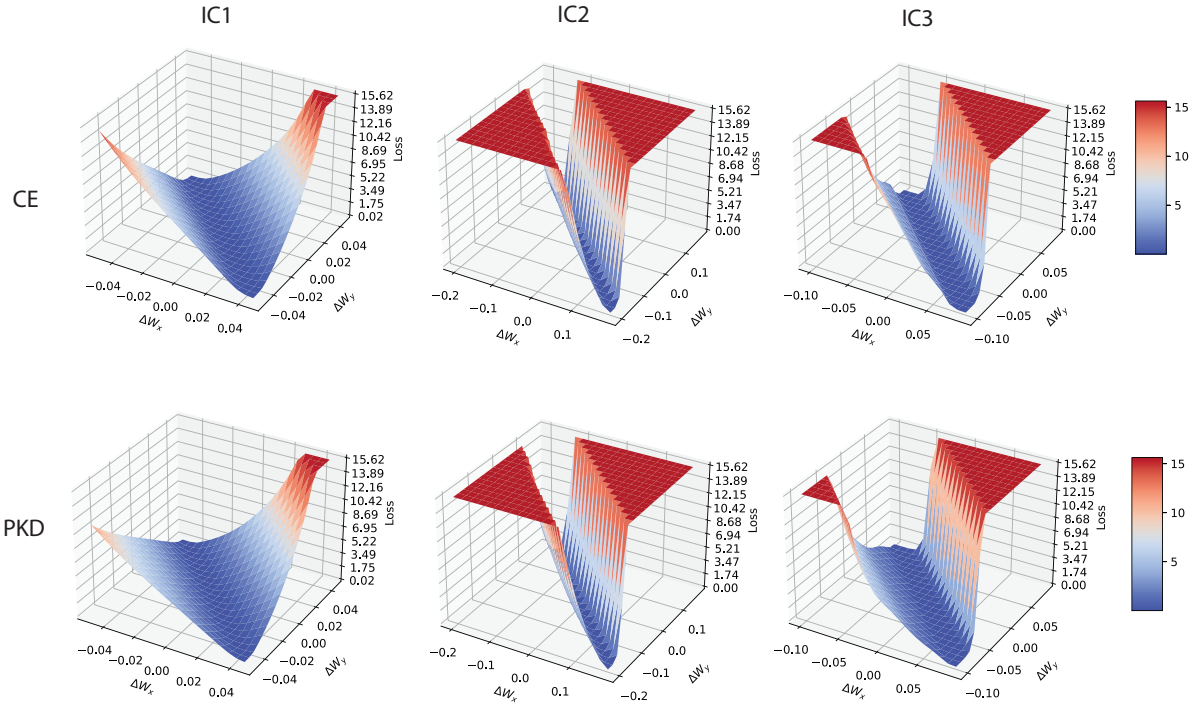


Figure 5.7. Visualizing the loss surface around the minimum of early exit classifiers attached to the I-frame backbone network when trained using CE vs PKD.

5.3.4 Evaluating WISE

We experimentally analyze the effectiveness of the proposed WISE methodology during inference. WISE has two key features: 1) the use of previous IC predictions (via lateral connections) available at the exit point to capture cross-modality information during inference, and 2) the efficient combination of these predictions using scaling factors. As a comparison, we evaluate WISE versus three alternative scenarios:

1. a baseline scenario not utilizing previous IC predictions (no lateral connections)
2. using previous ICs with uniform scaling factors
3. utilizing scaling factors obtained from WISE.

Note, that lateral connections at the L^{th} exit point mean the use of predictions from the previous $L-1$ ICs.

Experiment. For this experiment, we evaluate the three methods (no lateral connections, uniform scaling, WISE) and select a threshold for the exit points such that all the methods have iso-computational cost. Note when using lateral connections with uniform scaling, β is set to 1, that is all the previous predictions have equal weighting on the output. The accuracy of the three methods on UCF-101 and HMDB-51 datasets are shown in Table 5.2.

Results. The accuracy of each approach is summarized in Table 5.2. The results demonstrate the comparative accuracy of the baseline (no lateral connections), uniform scaling, and WISE scenarios. WISE outperforms the baseline and uniform scaling by $\approx 4.28\%$ and 0.8% respectively when using CoViAR as the backbone, and by $\approx 0.1\%$ and 4% using TEAM-Net as the backbone on UCF-101. Furthermore, WISE outperforms the baseline and uniform scaling by $\approx 9\%$ and 1% respectively when using CoViAR as the backbone, and by 0.4% and 4% using TEAM-Net as the backbone network on HMDB-51.

Table 5.2. Result Summary when using: 1) no lateral connections between the ICs, 2) uniform scaling factors and lateral connections between the ICs, 3) WISE. CoViAR and TEAM-Net are used as backbone networks. The reported accuracy is the average over the *three testing splits* and *three different seeds* (9 data points per entry) for UCF-101 and HMDB-51.

	Backbones			
	COVIAR		TEAM-Net	
Method	UCF-101	HMDB-51	UCF-101	HMDB-51
No Lateral Connections	88.59%	58.91%	93.24%	65.94%
Uniform Scaling (Ours)	92.08%	67.50%	89.49%	61.89%
WISE (Ours)	92.87%	68.21%	93.29%	66.29%

Table 5.3. Accuracy results for IC1, IC2, and IC3 when performing KD with I-frame only (I-frame KD), PKD curriculum, which performs KD between MV, then R and then I-frame (our proposal), and PKD anti-curriculum performs KD between I-frame, then R and then MV (also ours). The reported accuracy is the average over the three testing splits and over three different seeds for the UCF-101 and HMDB-51 datasets.

Dataset	Method	Backbones											
		CoViAR									TEAM-Net		
		MV			Residual			I-Frame			IC1	IC2	IC3
		IC1	IC2	IC3	IC1	IC2	IC3	IC1	IC2	IC3			
UCF-101	I-frame KD	34.51	47.19	57.38	46.70	60.92	73.25	48.93	59.41	78.00	9.71	30.80	49.52
	PKD curriculum	35.10	48.67	59.20	47.17	62.23	75.34	49.11	60.72	79.98	10.10	31.25	50.20
	PKD anti-curriculum	34.61	47.93	58.39	46.44	61.43	74.67	48.70	60.02	79.37	8.80	26.38	44.54
HMDB-51	I-frame KD	19.65	25.95	35.38	23.98	33.74	44.71	27.05	34.85	53.74	5.88	19.63	26.66
	PKD curriculum	20.10	27.43	38.08	24.77	35.27	48.07	27.32	35.15	53.84	5.99	19.81	26.78
	PKD anti-curriculum	19.54	26.24	36.21	24.61	35.33	47.30	26.78	34.14	52.68	5.57	16.32	23.12

5.3.5 Analysis of Teacher Classifier Order for PKD

Experiment. This analysis considers three approaches. Firstly, we evaluate a structured curriculum, termed PKD curriculum, which orders the FCs of the MV, R, and I-frame backbone networks (teacher network) from the highest to lowest accuracy. PKD curriculum is doing KD using MV, followed by R, and then I-frame FCs as teachers. Secondly, we evaluate the anti-curriculum PKD, starting with I-frame, followed by R, and then MV. Finally, we do KD using only the I-frame final classifier as the teacher. This analysis aims

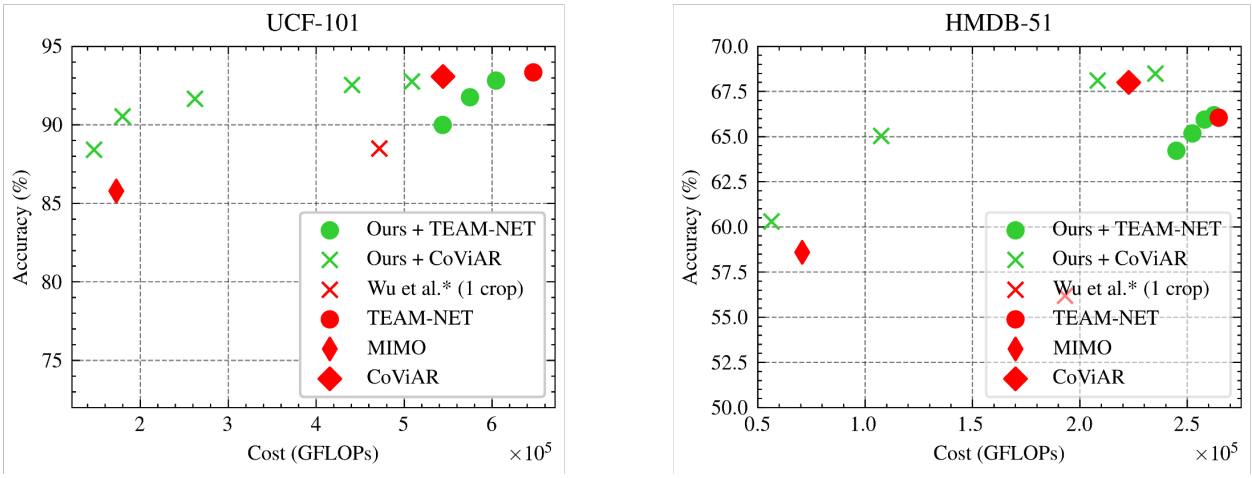


Figure 5.8. Trade-off curve of our proposal and SOTA work comparison on UCF-101 and HMDB-51. SOTA works (marked in red) fall either in the low accuracy and low computation cost regime (lower-left of the plot) or in the high accuracy and computation cost regime (top-right of the plot). Our proposal uses two backbones, i.e. CoViAR and TEAM-NET, (marked in green) scales in accuracy and computational cost.

to assess the efficacy of progressive knowledge transfer of the compressed video modalities against the standard approach of KD from a single, highly accurate teacher. To isolate the effect of the training order, we report results *without* using lateral connections. The results reported are the average over the three test splits and three seeds in Table 5.3 using 3 frames for each compressed video modality.

Results. From the results in Table 5.3, we observe that the PKD curriculum performs the best by up to 2% and 3% over I-frame KD on UCF-101 and HMDB-51 respectively. This observation aligns with insights from prior studies in the image classification domain [206], [207], underscoring that KD from the most accurate classifier may not always ensure the most efficient knowledge transfer.

5.3.6 Comparison with Prior Works

Experiment. We plot the trade-off curve between accuracy and computational cost (Giga Floating point operations - GFLOPs) in Figure 5.8 for UCF-101 and HMDB-51. We

Table 5.4. SOTA comparison (from top to bottom): 1) works using both CD and RD during training, 2) works using only CD but with high computational cost, 3) works focusing on optimizing the computational cost. CD: Compressed Domain, RD: Raw Domain, OF: Optical Flow.

Method	Modalites	Accuracy		GFLOPs	
		UCF-101	HMDB-51	UCF-101	HMDB-51
Liu et al. [192]	CD+RD	95.8%	73.5%	> 543,903	> 222,615
MFCD-Net [193]	CD+RD	93.2%	66.9%	1,328,536	543,764
CoViAR [190]	CD	93.1%	68.0%	543,903	222,615
CV-C3D [195]	CD	83.9%	55.7%	284,555	116,465
TEAM-NET [191]	CD	93.4%	66.1%	646,582	264,639
DMC-Net [194]	CD+OF	92.3%	71.8%	31,264,121	12,796,286
Wu et al. [199]	CD	88.5%	56.2%	4,717,060	1,930,655
MIMO [201]	CD	85.8%	58.6%	172,587	70,639
Ours	CD	**88.4%**	**60.3%**	**147,391**	**56,340**

report the cumulative GFLOPs for all three testing splits UCF-101 and HMDB-51. We compare with works that optimize for computational cost. Moreover, Table 5.4 summarizes the state-of-the-art (SOTA) methods: works using both Compressed Domain (CD) and Raw Domain (RD) during training (highlighted in blue), 2) works using only CD but with high computational cost (highlighted in orange), 3) works focusing on optimizing the computational cost (highlighted in green). Our proposal falls in the third category, and we compare it with these works in the trade-off plots (Figure 5.8).

Results. Figure 5.8 plots the accuracy-compute cost trade-off for UCF-101 and HMDB-51 of our proposal and SOTA methods. The trade-off is achieved by changing the threshold value τ of WISE. Note, that the threshold value is the same across all the ICs. We observe that MIMO [201] has a very low computational cost but trades off accuracy, while CoViAR [190] is at the other end of the spectrum where it trades off compute cost for accuracy. Our approach with PKD and WISE can efficiently scale between these two extremes by changing the threshold. Notably, our proposal showcases a reduction in GFLOPs under the iso-accuracy scenario, surpassing the efficiency of CoViAR [190], and the method proposed by Wu et al. [199]. On the other extreme when compared to MIMO [201] it achieves better

Table 5.5. Latency and Bandwidth comparison results.

Method	Latency	Bandwidth
MIMO [201]	8x	2.09x
Wu et al. [199]	3x	3.3x
CoViAR [190]	3x	3.3x
Our Method	1x	1x

accuracy at a similar compute cost for UCF-101 and significantly improved accuracy at the lower compute cost for the HMDB-51 dataset. Further, when compared with the MIMO [201] approach, our proposal has better latency and bandwidth efficiency, as reported in Section 5.3.7.

5.3.7 Latency and Bandwidth Results

We evaluate our proposal using the latency and bandwidth metrics. Consider the following scenario: video is transmitted in the form of packets that include Motion Vectors (MV), Residuals (R), and Intra-frames (I-frames). To classify a video stream, it is necessary to receive a sufficient number of these packets, as required by each classification technique. Therefore, we assess the latency based on the time interval required to classify a video stream at a consistent bit rate (i.e. iso-bandwidth). In our results, we provide relative latency between different SOTA methods for convenience.

To compare bandwidth effectively, we align the latency across different methods. Specifically, we determine how much bandwidth other techniques would require to match the latency of our method. This approach allows us to measure and compare the bandwidth efficiency of various techniques, ensuring they all meet the same latency target (i.e. iso-latency).

For this comparison, we use the CoViAR backbone with the proposed PKD methodology. CoViAR backbone uses a ResNet18 for MV, a ResNet18 for R, and a ResNet50 for I-frames. This allows the use of different numbers of frames for MV, R, and I-frames. For example, we can use 3 MV, 2 R, and 5 I-frames. In comparison, MIMO [201] uses a single network architecture, and this imposes the constraint of using the same number of MV, R, and I-

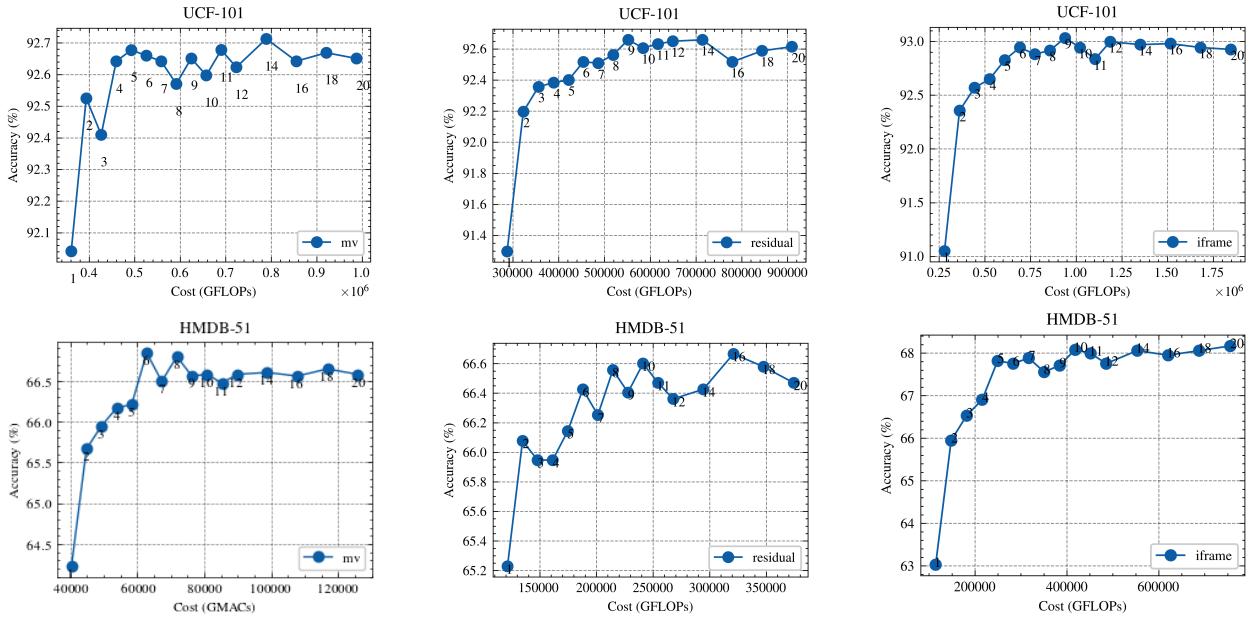


Figure 5.9. Study on the effect of number of frames when using our proposal with the CoViAR backbone when: 1) increasing the MV frames while keeping the number of R and I-frames constant, 2) increasing the R frames while keeping the number of MV and I-frames constant, and 3) increasing the I-frames while keeping the number of MV and R constant. The first row illustrates the plots for UCF-101 and the second row illustrates the plots for HMDB-51.

frames. The authors of MIMO [201] report results when using 8 MV, 8 R, and 8 I-frames in their experiments, and we follow the same when comparing with MIMO. The comparison results of our method with SOTA works for latency and bandwidth metrics are reported in Table 5.5. Notably, our proposal is 3x and 8x times more efficient than Wu et al. and MIMO in terms of latency, and 3.3x and 2.09x more efficient than Wu et al. and MIMO in terms of bandwidth. Please note that unlike the CoViAR backbone, the TEAM-Net backbone requires the same number of MV, R, and I-frames as input, and thus does not benefit from the proposed PKD methodology.

5.3.8 Effect of the Number of MV, R, and I-Frames

CoViAR [190] backbone employs a ResNet18 for MV, a ResNet18 for R, and a ResNet50 for I-frames. Thus, we explore the impact of using a different number of frames for each compressed video modality (MV, R, I-frame) on the accuracy and the computational cost (Giga Floating point operations - GFLOPs) during inference. TEAM-Net backbone requires the same number of MV, R, and I-frames as input. Therefore, this analysis cannot be done using our proposal with the TEAM-Net backbone.

Experiment. We perform the following experiments to isolate the effect of utilizing more frames on each compressed video modality:

1. We increase the number of MV frames while maintaining constant R and I-frames and observe how test accuracy changes (we use 3 R and 3 I-frames and the threshold is set to 99.99%).

2. Similarly we increase R frames while maintaining constant MV and I-frames (we use 3 MV and 3 I-frames and the threshold is set to 99.99%).

3. Similarly we increase I-frames while maintaining constant MV and R frames (we use 3 MV and 3 R frames and the threshold is set to 99.99%).

The accuracy was measured using the backbone networks (without the ICs) and the results were averaged over the three test splits. To calculate the GFLOP operations, we used the ptflops library [208].

Results. Figure 5.9 illustrates the effect of using more frames for each compressed video modality. Each plot in Figure 5.9 plots the computational cost (GFLOPs) on the x-axis and accuracy in percentage on the y-axis. Interestingly, beyond a certain number of frames, we observe a saturation point in accuracy despite increasing the GFLOPs. This underscores that an excessive number of frames fails to yield accuracy benefits.

5.4 Summary

In conclusion, this chapter presented a novel method for efficient compressed video action recognition, comprising two major components: PKD for training and WISE for inference. PKD uses the hierarchical nature of neural network convergence across the different video modalities: motion vectors, residuals, and intra-frames and facilitates the sequential transfer of knowledge leading to better generalization. The results of our experiments highlight the superiority of PKD over CE based training methods. Additionally, WISE further enhances inference accuracy by optimally combining outputs from the ICs. These findings underscore the potential of our proposed techniques to significantly advance the field of video action recognition, offering a promising avenue for future research and application.

6. SUMMARY AND FUTURE WORK

We conclude this thesis by underscoring the two primary objectives: privacy and efficiency in deep learning algorithms. The success of deep learning largely stems from the abundant availability of publicly available data. Using these datasets, we can train neural models to execute various machine learning tasks. However, certain data are private, preventing their public release. In such cases, it becomes imperative to find alternative ways to use these data, while maintaining privacy. Chapter 3 introduces a framework (DP-ImgSyn) that aims to address these challenges. Our proposal synthesizes images that can be publicly released instead of private data. These synthetic images have (ϵ, δ)-Differential Privacy guarantees, while being visually dissimilar from the private images. Additionally, they maintain similar utility as the private images. The synthetic images for neural network training yields comparable accuracy to the case of using private images. Therefore, with DP-ImgSyn, we can release synthetic images without compromising privacy.

Note, deep learning comes with significant time and computational overhead, requiring more efficient machine learning algorithms. One popular approach to reducing these demands is to quantize the weights and the activations of neural networks. However, determining the right bit width for quantization is difficult, due to the fact that the search space is exponential. In Chapter 4, we describe a method to determine the precision of weights and activations across the various layers of the neural network. Our technique uses a Multi-Layer Perceptron to predict the bit width for weights and activations in different layers, which reduces the network size and improves inference efficiency. Additionally, for action recognition tasks, we address efficiency by allowing the network to directly process compressed videos as inputs. Chapter 5 introduces a framework that gives compressed videos as input to the neural network, eliminating the need for video decompression. To further enhance efficiency, we incorporate early exits into neural networks and use progressive knowledge distillation.

Several future research directions can be sourced from this thesis. The effectiveness of the proposed privacy framework (DP-ImgSyn), evaluated on image classification tasks, could be explored in other tasks, such as object detection and scene understanding. Additionally, extending DP-ImgSyn to support multiple users sharing data with a central server (feder-

ated learning) raises interesting questions about communication costs, convergence steps, and network architecture diversity. For neural network quantization discussed in Chapter 4, examining the effects of different quantization methods is a promising area for future exploration. Lastly, studying the impact of various codecs, such as HEVC, on the classification performance could build upon the findings presented in Chapter 5.

REFERENCES

[1] F. D. Protection, "General data protection regulation (gdpr)," *Intersoft Consulting, Accessed in October*, vol. 24, no. 1, 2018.

[2] P. F. Edemekong, P. Annamaraju, and M. J. Haydel, "Health insurance portability and accountability act," 2018.

[3] E. Goldman, "An introduction to the california consumer privacy act (ccpa)," *Santa Clara Univ. Legal Studies Research Paper*, 2020.

[4] L. Sweeney, "Simple demographics often identify people uniquely," *Health (San Francisco)*, vol. 671, no. 2000, pp. 1–34, 2000.

[5] L. Sweeney, "Only you, your doctor, and many others may know," *Technology Science*, vol. 2015092903, no. 9, p. 29, 2015.

[6] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and trendső in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[7] D. Ravikumar, G. Saha, S. A. Aketi, and K. Roy, "Homogenizing non-IID datasets via in-distribution knowledge distillation for decentralized learning," *Transactions on Machine Learning Research*, 2024. [Online]. Available: https://openreview.net/forum?id=CuyJkNjIVd.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, Springer, 2006, pp. 265–284.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[10] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[13] J. Yang, X. Shen, J. Xing, *et al.*, "Quantization networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 7308–7316.

[14] S. Mei, X. Chen, Y. Zhang, J. Li, and A. Plaza, "Accelerating convolutional neural network-based hyperspectral image classification by step activation quantization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–12, 2021.

[15] R. M. Cichy and D. Kaiser, "Deep neural networks as scientific models," *Trends in cognitive sciences*, vol. 23, no. 4, pp. 305–317, 2019.

[16] C. M. Bishop, "Neural networks and their applications," *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[17] J. A. Anderson, *An introduction to neural networks*. MIT press, 1995.

[18] R. Kumari and S. K. Srivastava, "Machine learning: A review on binary classification," *International Journal of Computer Applications*, vol. 160, no. 7, 2017.

[19] Y.-C. Hsu, Z. Lv, J. Schlosser, P. Odom, and Z. Kira, "Multi-class classification without multi-class labels," *arXiv preprint arXiv:1901.00544*, 2019.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b*, vol. 4, no. 51-62, p. 56, 2017.

[22] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, 1988, pp. 21–28.

[23] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[24] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[25] Y. Li, C. Wei, and T. Ma, "Towards explaining the regularization effect of initial large learning rate in training neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[26] M. Moreira and E. Fiesler, "Neural networks with adaptive learning rate and momentum terms," 1995.

[27] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[28] A. G. Schwing and R. Urtasun, "Fully connected deep structured networks," *arXiv preprint arXiv:1503.02351*, 2015.

[29] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611–629, 2018.

[30] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[31] M. Sun, Z. Song, X. Jiang, J. Pan, and Y. Pang, "Learning pooling for convolutional neural network," *Neurocomputing*, vol. 224, pp. 96–104, 2017.

[32] H. Wu and X. Gu, "Max-pooling dropout for regularization of convolutional neural networks," in *Neural Information Processing: 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings, Part I 22*, Springer, 2015, pp. 46–54.

[33] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 371–380.

[34] X. Liu, W. Kong, and S. Oh, "Differential privacy and robust statistics in high dimensions," in *Conference on Learning Theory*, PMLR, 2022, pp. 1167–1246.

[35] N. Phan, X. Wu, H. Hu, and D. Dou, "Adaptive laplace mechanism: Differential privacy preservation in deep learning," in *2017 IEEE international conference on data mining (ICDM)*, IEEE, 2017, pp. 385–394.

[36] J. Dong, D. Durfee, and R. Rogers, "Optimal differential privacy composition for exponential mechanisms," in *International Conference on Machine Learning*, PMLR, 2020, pp. 2597–2606.

[37] L. Wasserman and S. Zhou, "A statistical framework for differential privacy," *Journal of the American Statistical Association*, vol. 105, no. 489, pp. 375–389, 2010.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pmlr, 2015, pp. 448–456.

[39] M. Awais, M. T. B. Iqbal, and S.-H. Bae, "Revisiting internal covariate shift for batch normalization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 11, pp. 5082–5092, 2020.

[40] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," *Advances in neural information processing systems*, vol. 31, 2018.

[41] D. Arpit, Y. Zhou, B. Kota, and V. Govindaraju, "Normalization propagation: A parametric technique for removing internal covariate shift in deep networks," in *International Conference on Machine Learning*, PMLR, 2016, pp. 1168–1176.

[42] Y. Cho, H. Cho, Y. Kim, and J. Kim, "Improving generalization of batch whitening by convolutional unit optimization," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 5321–5329.

[43] L. Huang, L. Zhao, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "An investigation into the stochasticity of batch whitening," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, 2020, pp. 6439–6448.

[44] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *Advances in neural information processing systems*, vol. 31, 2018.

[45] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," *arXiv preprint arXiv:1809.00846*, 2018.

[46] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," *Advances in neural information processing systems*, vol. 30, 2017.

[47] Y. Wu and J. Johnson, "Rethinking" batch" in batchnorm," *arXiv preprint arXiv:2105.07576*, 2021.

[48] C. Tian, Y. Xu, and W. Zuo, "Image denoising using deep cnn with batch renormalization," *Neural Networks*, vol. 121, pp. 461–473, 2020.

[49] E. S. Lubana, R. Dick, and H. Tanaka, "Beyond batchnorm: Towards a unified understanding of normalization in deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4778–4791, 2021.

[50] N. R. Goodman, "Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction)," *The Annals of mathematical statistics*, vol. 34, no. 1, pp. 152–177, 1963.

[51] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Micro-batch training with batch-channel normalization and weight standardization," *arXiv preprint arXiv:1903.10520*, 2019.

[52] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[53] C. Baskin, N. Liss, E. Schwartz, *et al.*, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 37, no. 1-4, pp. 1–15, 2021.

[54] G. J. Sullivan and T. Wiegand, "Video compression-from concepts to the h. 264/avc standard," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005.

[55] I. E. Richardson, *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[56] Z. Ahmed, A. J. Hussain, W. Khan, *et al.*, "Lossy and lossless video frame compression: A novel approach for high-temporal video data analytics," *Remote Sensing*, vol. 12, no. 6, p. 1004, 2020.

[57] N. D. Memon and K. Sayood, "Lossless compression of video sequences," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1340–1345, 1996.

[58] D. Le Gall, "Mpeg: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46–58, 1991.

[59] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[60] K. Sayood, *Lossless compression handbook*. Elsevier, 2002.

[61] V. Bhaskaran and K. Konstantinides, "Image and video compression standards: Algorithms and architectures," 1997.

[62] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[63] J.-K. Lee, N. Kim, S. Cho, and J.-W. Kang, "Deep video prediction network-based inter-frame coding in hevc," *IEEE Access*, vol. 8, pp. 95 906–95 917, 2020.

[64] Y. Nakaya and H. Harashima, "Motion compensation based on spatial transformations," *IEEE Transactions on circuits and systems for video technology*, vol. 4, no. 3, pp. 339–356, 1994.

[65] G. Strang, "The discrete cosine transform," *SIAM review*, vol. 41, no. 1, pp. 135–147, 1999.

[66] D. Zhang and D. Zhang, "Wavelet transform," *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval*, pp. 35–44, 2019.

[67] P. M. Bentley and J. McDonnell, "Wavelet transforms: An introduction," *Electronics & communication engineering journal*, vol. 6, no. 4, pp. 175–186, 1994.

[68] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of jpeg-2000," in *Proceedings DCC 2000. Data compression conference*, IEEE, 2000, pp. 523–541.

[69] G. Pastuszak and A. Abramowski, "Algorithm and architecture design of the h. 265/hevc intra encoder," *IEEE Transactions on circuits and systems for video technology*, vol. 26, no. 1, pp. 210–222, 2015.

[70] S.-M. Lei and M.-T. Sun, "An entropy coding system for digital hdtv applications," *IEEE transactions on circuits and systems for video technology*, vol. 1, no. 1, pp. 147–155, 1991.

[71] A. Moffat, "Huffman coding," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–35, 2019.

[72] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 620–636, 2003.

[73] N. Manjanaik, B. Parameshachari, S. Hanumanthappa, and R. Banu, "Intra frame coding in advanced video coding standard (h. 264) to obtain consistent psnr and reduce bit rate for diagonal down left mode using gaussian pulse," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 225, 2017, p. 012 209.

[74] B. Fischhoff, "Predicting frames," in *Judgment and Decision Making*, Routledge, 2013, pp. 184–206.

[75] B. Shen, I. K. Sethi, and B. Vasudev, "Adaptive motion-vector resampling for compressed video downscaling," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 6, pp. 929–936, 1999.

[76] B. Furht, J. Greenberg, and R. Westwater, *Motion estimation algorithms for video compression*. Springer Science & Business Media, 2012, vol. 379.

[77] S. Bachu and K. M. Chari, "A review on motion estimation in video compression," in *2015 International Conference on Signal Processing and Communication Engineering Systems*, IEEE, 2015, pp. 250–256.

[78] G. M. Schuster and A. K. Katsaggelos, "A video compression scheme with optimal bit allocation among segmentation, motion, and residual error," *IEEE Transactions on Image Processing*, vol. 6, no. 11, pp. 1487–1502, 1997.

[79] T. Ebrahimi and C. Horne, "Mpeg-4 natural video coding–an overview," *Signal Processing: Image Communication*, vol. 15, no. 4-5, pp. 365–385, 2000.

[80] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[81] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang, "Image and video compression with neural networks: A review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 6, pp. 1683–1698, 2019.

[82] S. Ponlatha and R. Sabeenian, "Comparison of video compression standards," *International Journal of Computer and Electrical Engineering*, vol. 5, no. 6, pp. 549–554, 2013.

[83] T. M. Hoang and J. Zhou, "Recent trending on learning based video compression: A survey," *Cognitive Robotics*, vol. 1, pp. 145–158, 2021.

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

[85] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

[86] D. D. Lewis, Y. Yang, T. Russell-Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

[87] J. Bennett, S. Lanning, *et al.*, "The netflix prize," in *Proceedings of KDD cup and workshop*, New York, vol. 2007, 2007, p. 35.

[88] M. Abadi, A. Chu, I. Goodfellow, *et al.*, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[89] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=HkwoSDPgg.

[90] A. Triastcyn and B. Faltings, "Generating artificial data for private deep learning," *arXiv preprint arXiv:1803.03148*, 2018.

[91] D. Ravikumar, E. Soufleri, A. Hashemi, and K. Roy, "Unveiling privacy, memorization, and input curvature links," *arXiv preprint arXiv:2402.18726*, 2024.

[92] D. Ravikumar, E. Soufleri, and K. Roy, *Curvature clues: Decoding deep learning privacy with input loss curvature*, 2024. arXiv: 2407.02747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2407.02747.

[93] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[94] M. Steiger, T. J. Bharucha, S. Venkatagiri, M. J. Riedl, and M. Lease, "The psychological well-being of content moderators: The emotional labor of commercial moderation and avenues for improving support," in *Proceedings of the 2021 CHI conference on human factors in computing systems*, 2021, pp. 1–14.

[95] P.-A. Goodfellow, X. Mirza, O. Warde-Farley, *et al.*, "Goodfellow i," *Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., Generative adversarial nets, Advances in neural information processing systems*, vol. 27, 2014.

[96] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," *arXiv preprint arXiv:1802.06739*, 2018.

[97] C. Xu, J. Ren, D. Zhang, Y. Zhang, Z. Qin, and K. Ren, "Ganobfuscator: Mitigating information leakage under gan via differential privacy," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2358–2371, 2019.

[98] X. Zhang, S. Ji, and T. Wang, "Differentially private releasing via deep generative model (technical report)," *arXiv preprint arXiv:1801.01594*, 2018.

[99] T. Cao, A. Bie, A. Vahdat, S. Fidler, and K. Kreis, "Dont generate me: Training differentially private generative models with sinkhorn divergence," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 480–12 492, 2021.

[100] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Hk4_qw5xe.

[101] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton, "Veegan: Reducing mode collapse in gans using implicit variational learning," *Advances in neural information processing systems*, vol. 30, 2017.

[102] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=B1xsqj09Fm.

[103] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[104] T. Salimans, H. Zhang, A. Radford, and D. Metaxas, "Improving GANs using optimal transport," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rkQkBnJAb.

[105] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1QRgziT-.

[106] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[107] E. Soufleri, D. Ravikumar, and K. Roy, "Dp-imgsyn: Dataset alignment for obfuscated, differentially private image synthesis," *Transactions on Machine Learning Research*,

[108] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[109] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3730–3738.

[110] Y. Long, B. Wang, Z. Yang, *et al.*, "G-pate: Scalable differentially private data generator via private aggregation of teacher discriminators," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2965–2977, 2021.

[111] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[112] J. C. Dressler, C. Bronk, and D. S. Wallach, "Exploiting military opsec through open-source vulnerabilities," in *MILCOM 2015-2015 IEEE Military Communications Conference*, IEEE, 2015, pp. 450–458.

[113] A. Act, "Health insurance portability and accountability act of 1996," *Public law*, vol. 104, p. 191, 1996.

[114] J. Manley and A. Cavoukian, "The personal information protection and electronic documents act (pipeda)," *Priv. gc. ca*, 2000.

[115] G. D. P. Regulation, "General data protection regulation (gdpr)," *Intersoft Consulting, Accessed in October*, vol. 24, no. 1, 2018.

[116] J. Deng, M. A. HALL-CRAGGS, D. Pellerin, *et al.*, "Real-time three-dimensional ultrasound visualization of erection and artificial coitus," *international journal of andrology*, vol. 29, no. 2, pp. 374–379, 2006.

[117] T. O. Abbas, M. AbdelMoniem, and M. E. Chowdhury, "Automated quantification of penile curvature using artificial intelligence," *Frontiers in Artificial Intelligence*, vol. 5, p. 954 497, 2022.

[118] R. Spence, A. Bifulco, P. Bradbury, E. Martellozzo, and J. DeMarco, "The psychological impacts of content moderation on content moderators: A qualitative study," *Cyberpsychology: Journal of Psychosocial Research on Cyberspace*, vol. 17, no. 4, 2023.

[119] F. C. Akyon and A. Temizel, "Deep architectures for content moderation and movie content rating," *arXiv preprint arXiv:2212.04533*, 2022.

[120] A. Bie, G. Kamath, and V. Singhal, "Private estimation with public data," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 653–18 666, 2022.

[121] A. Ganesh, M. Haghifam, M. Nasr, *et al.*, "Why is public pretraining necessary for private model training?" In *International Conference on Machine Learning*, PMLR, 2023, pp. 10 611–10 627.

[122] D. Chen, T. Orekondy, and M. Fritz, "Gs-wgan: A gradient-sanitized approach for learning differentially private generators," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 673–12 684, 2020.

[123] B. Wang, F. Wu, Y. Long, L. Rimanic, C. Zhang, and B. Li, "Datalens: Scalable privacy preserving training via gradient compression and aggregation," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2146–2168.

[124] J.-W. Chen, C.-M. Yu, C.-C. Kao, T.-W. Pang, and C.-S. Lu, "Dpgen: Differentially private generative energy-guided network for natural image synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8387–8396.

[125] T. Dockhorn, T. Cao, A. Vahdat, and K. Kreis, "Differentially private diffusion models," *arXiv preprint arXiv:2210.09929*, 2022.

[126] F. Harder, K. Adamczewski, and M. Park, "Dp-merf: Differentially private mean embeddings with randomfeatures for practical privacy-preserving data generation," in *International conference on artificial intelligence and statistics*, PMLR, 2021, pp. 1819–1827.

[127] S. Takagi, T. Takahashi, Y. Cao, and M. Yoshikawa, "P3gm: Private high-dimensional data release via privacy preserving phased generative model," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, IEEE, 2021, pp. 169–180.

[128] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, "Scalable private learning with pate," *arXiv preprint arXiv:1802.08908*, 2018.

[129] J. Jordon, J. Yoon, and M. Van Der Schaar, "Pate-gan: Generating synthetic data with differential privacy guarantees," in *International conference on learning representations*, 2019.

[130] C. Dwork, A. Roth, *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trendső in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[131] S. Gopi, Y. T. Lee, and L. Wutschitz, "Numerical composition of differential privacy," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 631–11 642, 2021.

[132] A. Yousefpour, I. Shilov, A. Sablayrolles, *et al.*, "Opacus: User-friendly differential privacy library in pytorch," in *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021. [Online]. Available: https://openreview.net/forum?id=EopKEYBoI-.

[133] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[134] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[135] *Imagenette - a subset of 10 easily classified classes from the imagenet dataset.* https://github.com/fastai/imagenette, 2018.

[136] F.-F. Li, A. Karpathy, and J. Johnson, *Tiny imagenet visual recognition challenge*, http://cs231n.stanford.edu/tiny-imagenet-200.zip, 2015.

[137] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[138] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3606–3613.

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

[140] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[141] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[142] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[143] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4829–4837.

[144] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5188–5196.

[145] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[146] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *Proceedings of the International Conference on Learning Representations (ICLR)*, ICLR, 2014.

[147] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[148] R. A. Haddad, A. N. Akansu, *et al.*, "A class of fast gaussian binomial filters for speech and image processing," *IEEE Transactions on Signal Processing*, vol. 39, no. 3, pp. 723–727, 1991.

[149] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[150] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[151] L. eon Bottou, "Online learning and stochastic approximations," *On-linelearning in neural networks*, vol. 17, no. 9, p. 142, 1998.

[152] T. Oshea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[153] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[154] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*, Springer, 2016, pp. 525–542.

[155] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, pp. 3123–3131, 2015.

[156] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[157] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *International Conference on Machine Learning*, 2017, pp. 3007–3016.

[158] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[159] W. Zhe, J. Lin, V. Chandrasekhar, and B. Girod, "Optimizing the bit allocation for compression of weights and activations of deep neural networks," in *2019 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2019, pp. 3826–3830.

[160] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7873–7882.

[161] Z. Liu, X. Zhang, S. Wang, S. Ma, and W. Gao, "Evolutionary quantization of neural networks with mixed-precision," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 2785–2789.

[162] Y. Yuan, C. Chen, X. Hu, and S. Peng, "Evoq: Mixed precision quantization of dnns via sensitivity guided evolutionary search," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.

[163] E. Soufleri and K. Roy, "Network compression via mixed precision quantization using a multi-layer perceptron for the bit-width allocation," *IEEE Access*, vol. 9, pp. 135 059–135 068, 2021.

[164] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[165] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[166] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[167] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4300–4309.

[168] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[169] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[170] A. Kosta, E. Soufleri, I. Chakraborty, A. Agrawal, A. Ankit, and K. Roy, "Hyperx: A hybrid rram-sram partitioned system for error recovery in memristive xbars," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 88–91.

[171] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[172] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," *arXiv preprint arXiv:1705.01462*, 2017.

[173] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," *arXiv preprint arXiv:2003.11535*, 2020.

[174] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178.

[175] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*, PMLR, 2016, pp. 2849–2858.

[176] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[177] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[178] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*, IEEE, 1993, pp. 293–299.

[179] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[180] I. Garg, P. Panda, and K. Roy, "A low effort approach to structured cnn design using pca," *IEEE Access*, vol. 8, pp. 1347–1360, 2019.

[181] S. Roy, P. Panda, G. Srinivasan, and A. Raghunathan, "Pruning filters while training for efficiently optimizing deep learning networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–7.

[182] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[183] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[184] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *international conference on machine learning*, PMLR, 2017, pp. 2847–2854.

[185] A. Ankit, I. El Hajj, S. R. Chalamalasetti, *et al.*, "Panther: A programmable architecture for neural network training harnessing energy-efficient reram," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1128–1142, 2020.

[186] H. Darvishi, D. Ciuonzo, E. R. Eide, and P. S. Rossi, "Sensor-fault detection, isolation and accommodation for digital twins via modular data-driven architecture," *IEEE Sensors Journal*, vol. 21, no. 4, pp. 4827–4838, 2020.

[187] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[188] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet of Things Journal*, 2021.

[189] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao, "Video coding for machines: A paradigm of collaborative compression and intelligent analytics," *IEEE Transactions on Image Processing*, vol. 29, pp. 8680–8695, 2020.

[190] C.-Y. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krähenbühl, "Compressed video action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6026–6035.

[191] Z. Wang, Q. She, and A. Smolic, "Team-net: Multi-modal learning for video action recognition with partial decoding," *arXiv preprint arXiv:2110.08814*, 2021.

[192] Y. Liu, J. Cao, W. Bai, B. Li, and W. Hu, "Learning from the raw domain: Cross modality distillation for compressed video action recognition," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023, pp. 1–5.

[193] B. Battash, H. Barad, H. Tang, and A. Bleiweiss, "Mimic the raw domain: Accelerating action recognition in the compressed domain," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 684–685.

[194] Z. Shou, X. Lin, Y. Kalantidis, *et al.*, "Dmc-net: Generating discriminative motion cues for fast compressed video action recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1268–1277.

[195] S. F. dos Santos, N. Sebe, and J. Almeida, "Cv-c3d: Action recognition on compressed videos with convolutional 3d networks," in *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, IEEE, 2019, pp. 24–30.

[196] S. Hochreiter and J. Schmidhuber, "Simplifying neural nets by discovering flat minima," *Advances in neural information processing systems*, vol. 7, 1994.

[197] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural computation*, vol. 9, no. 1, pp. 1–42, 1997.

[198] E. Soufleri, D. Ravikumar, and K. Roy, *Advancing compressed video action recognition through progressive knowledge distillation*, 2024. arXiv: 2407.02713 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2407.02713.

[199] M.-C. Wu, C.-T. Chiu, and K.-H. Wu, "Multi-teacher knowledge distillation for compressed video action recognition on deep neural networks," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 2202–2206.

[200] M. Havasi, R. Jenatton, S. Fort, *et al.*, "Training independent subnetworks for robust prediction," *arXiv preprint arXiv:2010.06610*, 2020.

[201] H. Terao, W. Noguchi, H. Iizuka, and M. Yamamoto, "Efficient compressed video action recognition via late fusion with a single network," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023, pp. 1–5.

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

[203] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: A large video database for human motion recognition," in *2011 International conference on computer vision*, IEEE, 2011, pp. 2556–2563.

[204] J. Lin, C. Gan, and S. Han, "Tsm: Temporal shift module for efficient video understanding," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 7083–7093.

[205] W. Kay, J. Carreira, K. Simonyan, *et al.*, "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[206] S. You, C. Xu, C. Xu, and D. Tao, "Learning from multiple teacher networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1285–1294.

[207] G. Kaplun, E. Malach, P. Nakkiran, and S. Shalev-Shwartz, "Knowledge distillation: Bad models can be good role models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 683–28 694, 2022.

[208] V. Sovrasov. "Ptflops: A flops counting tool for neural networks in pytorch framework." (2018-2023), [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch.