MSc Artificial Intelligence
Master Thesis

Efficient Quantized Vision Transformers for Deepfake Detection

by
Ingur Veken
11886366

August 27, 2024

48 EC
2023/2024

Supervisor:
Dr. F. Regazzoni

Examiner:
Dr. G. Sileno

Second reader:
Dr. F. Regazzoni

Universiteit van Amsterdam

Contents

Abstract

1 Introduction

Recent advances in Artificial Intelligence (AI) and deep learning have led to considerable progress in the field of computer vision and image synthesis. In particular, these advancements have greatly improved the quality of deepfakes - AI techniques that enable realistic-looking face swapping in images and videos. However, while this technology has shown promise in various creative applications, such as in movie localization and special effects, it has also raised significant concerns. The widespread ability to manipulate videos, making individuals appear to say or do things they never said or did, poses severe threats to privacy, security, and trust in digital media [15]. For this reason, deepfake detection methods have become critical to protect those susceptible to this technology.

Early attempts to detect deepfakes predominantly used Convolutional Neural Networks (CNNs), due to their successful application in various computer vision tasks. Various CNN architectures showed promising results on first generation deepfake datasets [1, 16, 33, 68]. However, as the quality, quantity, and diversity of these datasets increased, more powerful models need to be developed.

In 2020, computer vision research began shifting towards Transformer-based models following its success in large language models (LLMs) for Natural Language Processing (NLP). Transformers quickly became the de facto standard architecture in NLP, as the self-attention mechanism allowed for capturing long-range dependencies and complex contexts [75]. The scaling successes of Transformers led to the development of the Vision Transformer architecture, which uses raw image patches rather than text tokens as input [26]. To address the lack of inductive biases found in CNN architectures, these models largely relied on larger datasets for pre-training. This allowed Vision Transformers to achieve performance comparable to state-of-the-art CNN models, sometimes even outperforming them [74].

In deepfake detection, ViT-based architectures either use Transformers after a CNN to further refine the extracted features, or replace the CNN entirely. Nonetheless, the adoption of Vision Transformers as feature extractors for deepfake detection still faces various challenges. As ViTs continue to improve and grow in size, the increasing computational and memory requirements create a significant bottleneck. This is especially noticeable in resource-constrained environments, such as for edge devices or real-time applications [27].

Recently, different approaches to model compression have been researched, including quantization, knowledge distillation, and pruning [53, 73, 85]. Quantization maps high precision model weights to lower bit-widths without changing the model architecture, and is regarded as one of the most effective approaches to model compression [30]. Most quantization methods are done post-training (PTQ), as this does not require any changes to existing training pipelines, reduces the memory footprint and can speed up model inference drastically [51]. Unfortunately, these methods also result in accuracy loss, especially at lower precisions [22, 76].

Quantization-Aware Training (QAT) aims to counter this loss in accuracy by emulating quanti-

zation during training [76]. This way, the model learns to adapt to the reduced precision, which allows it to maintain higher accuracy at inference-time at the cost of being more time-consuming and resource intensive to train [27].

More recently, BitNet introduced the BitLinear layer, a custom linear layer that uses ternary $\{-1, 0, +1\}$ weights, designed for scalable and stable Transformer LLMs. BitNet replaces all linear layers with BitLinear layers that are trained using QAT, to significantly reduce memory and energy consumption compared to their higher-precision counterparts [56, 76].

Motivated by the need for more accessible and efficient detection models, the goal of this thesis is to investigate the application of BitLinear layers to Vision Transformers for deepfake detection. Specifically, this thesis attempts to answer the following research questions:

- How can Vision Transformers be efficiently adapted for deepfake detection in resource-constrained environments?
- What is the impact of using BitLinear layers in Vision Transformers on the model size, speed, and detection performance?
- How do quantization-aware trained models compare to post-training quantized models in the context of deepfake detection?

To address these questions, we aim to make the following contributions TODO: rewrite these:

- We implement the BitLinear layer for training custom networks, based on the BitNet paper.
- We pre-train two Vision Transformers, a baseline using full-precision linear layers and our implementation of BitLinear layers, on ImageNet-1k.
- We fine-tune these models on multiple deepfake datasets, FF++, Celeb-DF (and maybe a DFDC subset), and compare the results.
- We quantize the baseline model to compare the effectiveness of the QAT and PTQ approach.

The remainder of this thesis will be structured as follows: TODO: briefly explain structure

2 | Literature review

In this chapter, we discuss past research that focuses on deepfake detection methods, CNN- and Transformer-based architectures, and challenges. We highlight differences and similarities among models, and the problems they address.

2.1 Datasets

Multiple datasets have been released over the past few years to support the advancement of deepfake detection models. These datasets can be categorized into three generations based on their release date, size, and synthesis quality. The first generation consists of small datasets such as UADFV [80] and DF-TIMIT [44], as well as FaceForensics++ (FF++) [61], which provided the initial benchmarks for early deepfake detection models. Of these datasets, FF++ is the largest and most widely explored dataset, containing 1000 real and 4000 counterfeit videos. The second generation dataset includes Google DFD [29], DFCD-Preview [24], and Celeb-DF [50], and greatly improved the visual quality of the deepfake videos, providing more challenging data for detection models. Celeb-DF is the most used dataset of this generation, offering 590 real and over 5000 fake videos, corresponding to more than 2 million unique frames. Finally, the third generation consists of the DeepFake Detection Challenge (DFDC) [25] and Deeper Forensics [42] datasets, increasing the amount of data and variety even further. Most importantly, these datasets address the most common problem with the first two generations, namely the limited amount of swapped identities [25]. For example, Celeb-DF uses videos from 59 different celebrities, whereas DFDC contains 100,000 videos of 3426 paid actors, which amounts to tens of millions of frames.

2.2 Convolutional Networks

Fundamentally, deepfake detection is an image classification problem, where models need to be able to detect slight differences between fake and real faces. For this reason, initial deep learning research in deepfake detection used Convolutional Neural Networks (CNNs) to identify manipulated images/videos, due to their widespread success in image classification tasks [8].

In 2018, MesoNet [1] presented one of the first deep learning approaches to deepfake detection. MesoNet uses a very compact CNN architecture designed for efficiency and focuses on detecting mesoscopic (mid-level) features in images. The authors also introduced MesoInception, which uses inception modules to capture multiscale features to further increase its detection methods [67].

In contrast, Li and Lyu [49] used large conventional CNN models to detect specific warping artifacts observed in early deepfake videos. This work took advantage of the computational limitations of early deepfake algorithms, which only synthesized face images of fixed sizes, and

applied affine warping to match the target faces. The warping process results in visible artifacts, as a result of the resolution inconsistency between the warped face area and surrounding context. The authors experimented with four different CNN models based on the VGG16, ResNet50, ResNet101, and ResNet152 architectures [33, 63]. These models were trained using a self-generated dataset, whereas evaluation was done on the UADFV and DF-TIMIT datasets. Furthermore, they also trained and compared their results to the MesoNet and MesoInception models on the same datasets. Their results showed that the ResNet50 model outperformed all other models on both datasets.

Wang et al. [77] combined the VGGFace face detection model [9] with ResNet50 as backend architecture for their model. Their approach monitors the internal activations of the neurons at various layers and captures statistical data, which is then used to train the classifier. This model achieved AUC scores of 98.5% on FF++, 66.8% on Celeb-DF, and 68% on the DFDC dataset.

Another study compared selecting the entire face as input to detection models versus only selecting specific regions [71]. Here, the XceptionNet [16] architecture was used to create the detection models, and the results were evaluated on datasets from both the first (UADFV, FF++) and second generation (Celeb-DF, DFDC-Preview). Evaluation showed significant improvements in realism for specific facial regions between the first and second generation datasets, especially the nose, mouth, and edge of face regions. The results also show that full face input models outperform all specific facial region based models across all datasets.

For the DeepFake Detection Challenge [25], the authors performed a meta analysis of the top submissions to the detection challenge. The top-5 winning solutions all used ensemble learning, which combines multiple model outputs to further increase the accuracy. Moreover, frame-by-frame classification models also generally performed better when compared to multi-frame models. The first-place submission used MTCNN [84] for face detection and an ensemble of seven models based on the EfficientNet B7 [68] architecture for feature encoding. Additionally, the training made heavy use of different augmentations and also included dropping structured parts of faces to further improve generalizability [21]. The second solution was based on the XceptionNet and EfficientNet B3 architecture and a WS-DAN model [38] for augmentation. The third place submission also used an ensemble of EfficientNet models and made use of the MixUp augmentation [83].

2.3 Vision Transformers

The modern Transformer architecture was introduced in 2017 by Vaswani et al. [75] and primarily designed for Natural Language Processing (NLP) tasks. The self-attention mechanism allows for capturing long-range dependencies and understanding complex contexts. However, while Transformers quickly became the de facto standard for NLP tasks, this design initially saw limited application in Computer Vision tasks. This is because naive application of self-attention on high-dimensional data such as images would be too computationally and memory intensive, as the cost of self-attention scales quadratically ($O(n^2)$) with the dimensionality of the data. Additionally, Transformers lack some of the inductive biases found in CNN architectures, including locality, and translation equivariance [26].

Despite these initial limitations, Dosovitskiy et al. [26] introduced the Vision Transformer (ViT) in 2020 after the scaling successes of Transformers in NLP. ViT models work by splitting images into linearly embedded fixed-sized patches and considering each patch as a token in a sequence (see Fig. 2.1). This patch-based approach allows for more efficient application of the

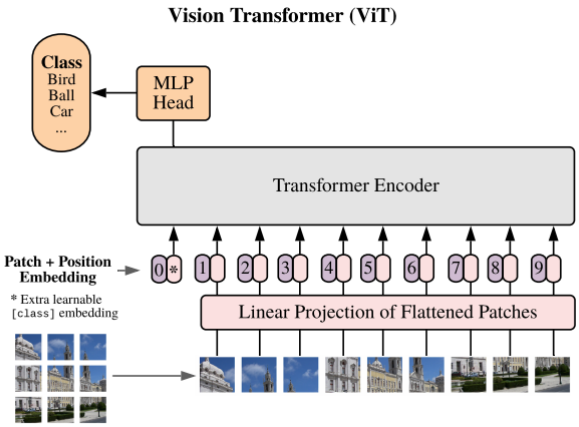

Figure 2.1: Vision Transformer (ViT) architecture. Images are split into fixed-sized patches, which are linearly projected. Positional encoding is applied to the resulting sequence of projected patches, which are then processed by a transformer encoder. Figure from [26].

self-attention mechanism, whilst still capturing complex relationships between different parts of the image. To address the lack of inductive biases found in CNN architectures, ViT models rely on pre-training with large datasets. This allows the architecture to achieve performance comparable to state-of-the-art CNNs on multiple image recognition benchmarks.

In the context of deepfake detection, Vision Transformers are mainly used in two ways: following a CNN or as a replacement for CNNs as the main feature extractor [58]. Wodajo and Atnafu [78] introduced the CViT model which uses a CNN model based on the VGG architecture [63] as feature learning component. This is motivated by the fact that CNNs excel at learning local features, while Transformers can learn from local and global feature maps. The combined capacity enables their model to better correlate relationships between non-local features.

Later, Coccomini et al. [17] expanded upon this architecture by replacing the VGG-CNN with an EfficientNet B0 as the local feature extractor. The authors also replace the ViT with a CrossViT based multiscale architecture [13]. CrossViT uses a dual-branch Transformer approach that processes both small and large patch tokens in two separate branches and uses cross-attention to exchange information between the different scales. This allows the model to better detect artifacts introduced by deepfakes both on a local and global level.

In contrast, Heo et al. [36] used a Data-Efficient Image Transformer (DeiT) [73] as the main feature extractor. DeiT uses a student-teacher strategy and introduces a special distillation token. In this case, an EfficientNet B7 is used to produce soft labels, which the ViT student model learns to replicate. This method provides an extra learning signal to the student model, which speeds up the training and further increases the accuracy of the model, allowing it to be trained using less data.

Recent work also investigated the efficacy of self-supervised learning (SSL) on pre-trained Vision Transformers for deepfake detection [58]. Self-supervised learning is a method for learning robust representations from unlabeled data, and played an important role in the success of Transformers in NLP. In language modelling, self-supervised training objectives provide a more complex learning signal than supervised objectives of a single label per sentence [60]. Similarly, Caron et al. [10] argue that image-level supervision often reduces the rich visual information contained in an image to a single concept selected from a predefined category set. The authors applied SSL to DeiT, which led to the development of DINO, extending knowledge distillation

to work without explicit labels. Nguyen et al. [58] show that fine-tuning DINO for deepfake detection can lead to superior performance compared to fine-tuning models that were trained in a supervised manner.

2.4 Quantization

As Transformer-based architectures continue to scale in size and capabilities, so do the memory and computational requirements. This introduces significant challenges when deploying these models for real-life applications, especially in resource constrained environments [27]. For this reason, various approaches to model compression have been explored, including knowledge distillation (DeiT) [73], weight pruning [85], low-rank factorization [12], and quantization [53]. Among these techniques, quantization stands out due to its consistent success in both training and inference of AI models and can significantly reduce the memory footprint and speed up model inference [30].

Quantization works by mapping high precision model weights to lower bit-widths without having to change the underlying model architecture. This is especially useful for Transformer-based architectures, which are often carefully designed. Furthermore, quantization also stimulates the development of more efficient hardware design utilizing lower-bit data [18, 40, 76].

Most quantization methods are applied post-training (PTQ), which does not require any changes to the existing training pipeline. PTQ directly quantizes the model weights without requiring re-training or fine-tuning, making it the current preferred choice for expensive large-scale models [51]. In practice, however, these methods also suffer from significant accuracy loss, especially for ultra low-precision quantization (≤ 4 bits) [22, 27]. This performance degradation can be attributed to the significant difference between the original and quantized weights, for which the model is not optimized during training [41, 76]. And while some sub 8-bit PTQ frameworks such as GPTQ [28], QuIP [11], RepQuant [51], and SmoothQuant+ [59] have shown promising early results in mitigating some of the performance degradation, it is still an active area of research.

Alternatively, Quantization-Aware Training (QAT) aims to alleviate the loss in accuracy by emulating the quantization during the training phase [27]. Models trained using QAT learn to adapt to the reduced precision right away, allowing these models to retain higher accuracy at inference-time. The main challenges of QAT lie in the gradient approximations for the non-differentiable quantization operations. Moreover, the increased performance of QAT at inference-time comes at the cost of being more time-consuming and resource intensive during training. This is because QAT requires retraining the model from scratch, and takes longer for lower precision models [22, 27].

In addition to lower memory consumption and faster inference, low-level binary $\{-1, +1\}$ and ternary $\{-1, 0, +1\}$ quantization allows us to simplify matrix multiplications. Specifically, the expensive floating point multiplications can be replaced with efficient bitwise or addition operations, greatly improving the power efficiency of the model [86]. In this domain, Spiking Neural Networks quantize the activations [57, 87], whereas Binarized and Ternary Neural Networks (BNNs and TNNs) quantize the weights [3, 18].

For vision tasks, architectures such as BiViT [34] and BinaryViT [46] successfully applied model binarization to Vision Transformers. However, while these models outperform previous binarization techniques, they still suffer from noticeable performance degradation.

In language modeling, BitNet demonstrated the scalability of 1-bit quantization in Transformer LLMs [76]. Specifically, the authors introduce the BitLinear layer, which combines 8-bit activation quantization with binary weights that are trained using QAT. BitNet is based on the successful LLaMA architecture [72], only replacing all linear layers with BitLinear layers. When compared to state-of-the-art PTQ methods [11, 28, 79], BitNet achieves comparable performance to 8-bit precision and outperforms all lower precision quantization methods.

More recently, Ma et al. [56] introduced BitNet b1.58, replacing the binary weights with ternary weights. The authors show that this approach retains all the benefits of the original BitNet, while further increasing its performance, computational efficiency, and memory usage by introducing sparsity in the model. Additionally, the results show that BitNet b1.58 can match full precision baselines when scaled sufficiently. For this reason, exploring the BitLinear layer applied to modern ViT architectures could lead to more efficient and accessible models, while maintaining high classification accuracy.

3 | Methodology

In this chapter, we discuss our proposed architectures for this thesis. We refer to our base modified Vision Transformer architecture as LinViT, and the BitLinear-based variant as BitViT. We first introduce our data preprocessing pipelines on the ImageNet-1k classification dataset [62], the FaceForensics++ [61] and Celeb-DF [50] deepfake detection datasets. Next, we discuss the Vision Transformer architecture, as well as our proposed modifications for LinViT, in Sec. 3.2. For our BitViT model, we replace all linear layer modules in the transformer encoder of LinViT with our implementation of BitLinear as proposed by Wang et al. [56, 76], which we describe in detail in Sec. 3.3. Then, in Sec. 3.4 we integrate knowledge distillation in our training pipeline as proposed by Touvron et al. [73] to allow for more data-efficient training. Finally, in Sec. 3.5 we formulate different methods for post-training quantization for LinViT to compare against the effectiveness of QAT in BitViT. The complete PyTorch [4] code implementation can be found on `https://github.com/ingur/` link

3.1 Data Preprocessing

We pretrain our custom Vision Transformer on ImageNet-1k [62]. The original ViT paper focused on large-scale pre-training on ImageNet-21k and JFT-300M [66], which require significant computational resources. However, recent advancements [47, 73] allow us to train more efficiently on mid-sized datasets, including ImageNet-1k, greatly reducing the cost of pre-training. After pre-training, we fine-tune our proposed model on two deepfake datasets, FaceForensics++ [61] and Celeb-DF [50]. These datasets are widely explored datasets, which makes it easier to compare architectural changes when training for these datasets [70]. Due to computational and resource constraints, we decided to only consider these two mid-size deepfake datasets, and leave third generation datasets (e.g. DFDC [25]) for future work. An overview of the used datasets can be found in Tab. 3.1.

ImageNet. The ImageNet-1k dataset was obtained through the HuggingFace [1] datasets library [48]. As this dataset contains multiple image modes, we first ensure all images follow a consistent *RGB* format using the `Image.convert()` method from the Pillow library. We also resize the images so that the smaller edges are always 256 pixels, while preserving the aspect ratio. This is done using the `torchvision.transforms.Resize()` method from PyTorch [4].

Deepfake datasets. The FaceForensics++ and Celeb-DF dataset are both available through their respective GitHub repositories [2] [3], and require access to be requested through a form. Both datasets consist of multiple directories of videos, and share a common preprocessing pipeline. We extract an equal number of frames from both the real and fake videos using a 30-frame interval. For FaceForensics++, we processed 1000 real and 1000 fake videos. For Celeb-DF,

[1] `https://huggingface.co/datasets/ILSVRC/imagenet-1k`
[2] `https://github.com/ondyari/FaceForensics`
[3] `https://github.com/yuezunli/celeb-deepfakeforensics`

we used all 890 real and randomly sampled an equal amount of fake videos. Next, we use the MTCNN face detection model to detect faces in each frame, and discard frames without faces. We retain only the largest faces detected in each frame and ensure the model focuses on the most likely faces, and discard the remaining ones. Next, each face is returned as a 256x256 cropped image with 40-pixel margins. Finally, we split the processed data into train, validation and test sets using an 80/10/10 ratio. These subsets allow us to evaluate the model's ability to adapt to deepfake detection using limited examples.

Dataset	#Train images	#Validation images	#Test images
ImageNet-1k	1,281,167	50,000	100,000
FaceForensics++	27,940	3,508	3,414
Celeb-DF-V2	19,940	2,390	2,334

Table 3.1: Overview of the datasets used for the experiments. For FaceForensics++ and Celeb-DF-V2, a balanced subset of the data is used for fine-tuning.

Augmentation. Data augmentation is a popular method to reduce overfitting and improve generalization of deep learning models, done by generating additional training data [19]. Simple transformations such as horizontal flips and random cropping are commonly used in image classification or deepfake detection models [25, 45]. Our pre-training mainly relies on three recent augmentation techniques: RandAugment [19], MixUp [83] and CutMix [81].

RandAugment was introduced by Cubuk et al. [19] and can be used to apply a fixed number of random augmentation operations to images. Additionally, the strength and number of the applied augmentations can be controlled with two simple parameters. This results in a simple and consistent pipeline that has shown to outperform many complex augmentation strategies without extensive need of hyperparameter tuning.

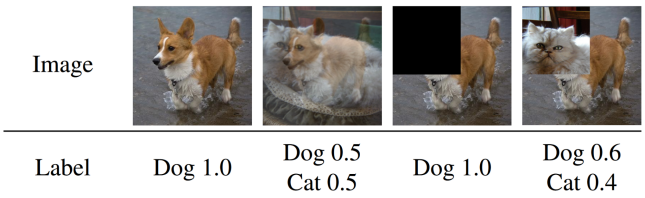

Figure 3.1: Comparison of various image augmentation techniques, from left to right: original image, Mixup [83], Cutout [23] and CutMix [81]. Figure from [81].

Next, MixUp and CutMix are used to create inter-class examples (see Fig. 3.1). MixUp constructs new training examples by taking linear interpolations of random pairs of images and their corresponding labels [83]. Given two training examples: (x_i, y_i) and (x_j, y_j), this method creates a new training example $(\tilde{x}, \tilde{y})$ using a mixing parameter λ from a beta distribution:

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1 - \lambda) x_j \\ \tilde{y} &= \lambda y_i + (1 - \lambda) y_j \end{aligned} \tag{3.1}$$

This method extends the training distribution and enhances model robustness and performance against adversarial examples.

Similarly, CutMix is another method that combines two image samples, and linearly interpolates the labels [81]. In contrast to CutOut [23], this method replaces a random image region with a patch from another training example, preventing inefficiency during training and leading to better localization capabilities.

3.2 Vision Transformer

This section describes our base Vision Transformer architecture. The proposed architecture builds upon the original ViT as proposed by Dosovitskiy et al. [26]. We apply minor simplifications and optimizations based on the work of various other research projects [7, 47, 65, 82]. We first describe the original ViT architecture, and then explain our modifications. We call the modified Vision Transformer architecture LinViT.

3.2.1 Patch embedding and positional encoding

The standard Transformer architecture is designed to receive a 1D sequence of D-dimensional token embeddings z_0 as input [75]. Let $x \in \mathbb{R}^{H \times W \times C}$ be a 2D input image[4], where (H, W) denote the height and width, and C the number of channels. First, the input image is divided into non-overlapping flattened patch vectors:

$$\mathcal{P}(x) = [x_p^1, \ldots, x_p^N] \qquad x_p \in \mathbb{R}^{N \times (P^2 \cdot C)} \tag{3.2}$$

Here, P indicates the patch size and $N = HW/P^2$ the resulting number of patches. Next, patch embeddings can be obtained through layer normalization (LN) [5] followed by a linear projection of each patch into the space of the transformer encoder. This process is called tokenization ($\mathcal{T}$) and is defined as follows:

$$\mathcal{T}(x) = \mathrm{LN}(\mathcal{P}(x))\mathbf{E}_t = [x_p^1\mathbf{E}_t, \ldots, x_p^N\mathbf{E}_t] \qquad \mathbf{E}_t \in \mathbb{R}^{(P^2 \cdot C) \times D} \tag{3.3}$$

Here, $\mathbf{E}_t$ is the learnable embedding matrix for tokens, and D is the hidden dimensionality of the Transformer. Next, a special learnable classification token x_{class} is prepended to the embedded patches, which serves as a global representation of the entire image for classification tasks. Finally, learnable positional embeddings $\mathbf{E}_{pos}$ are added to retain positional information. The resulting sequence z_0 is then used as the input to the first transformer encoder:

$$z_0 = [x_{\text{class}}, \mathcal{T}(x)] + \mathbf{E}_{pos} \qquad x_{\text{class}} \in \mathbb{R}^D,\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{3.4}$$

Shifted Patch Tokenization. We extend our tokenization by using the Shifted Patch Tokenization (SPT), which utilizes spatial relations between neighboring pixels during tokenization to capture more spatial information [47]. While originally introduced for ViT training on small-size datasets, SPT has also been found to improve ViTs even on mid-size datasets such as ImageNet-1k. SPT works by shifting the input image in four diagonal directions, creating four permutations of the input image. These permutations are then concatenated to the original image before being divided into non-overlapping patches and tokenized. This process is illustrated in Fig. 3.2 and is formulated as follows:

$$\mathrm{SPT}(x) = \mathrm{LN}(\mathcal{P}([x, x_s^1, \ldots, x_s^{N_S}]))\mathbf{E}_S \qquad \mathbf{E}_S \in \mathbb{R}^{(P^2 \cdot C \cdot (N_S+1)) \times D} \tag{3.5}$$

where x_s^i represents the i-th shifted image, N_S the number of image shifts and $\mathbf{E}_S$ the new learnable linear projection.

Positional Encoding. The Vision Transformer architecture as introduced by Dosovitskiy et al. [26] uses learnable positional embeddings $\mathbf{E}_{pos}$. We replace these embeddings with fixed

[4]To simplify our formulations, we ignore the batch dimension.

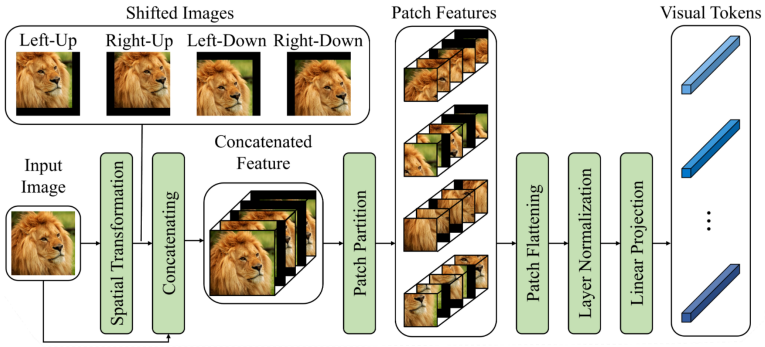

Figure 3.2: Overview of Shifted Patch Tokenization (SPT). Shifted permutations of the input image are concatenated before being transformed to patch embeddings (visual tokens). Figure from [47].

sinusoidal 2D positional encodings to reduce the amount of trainable parameters. This approach extends the 1D sinusoidal encodings introduced in the original Transformer architecture [75] to 2D, and provides a simple representation of position with comparable performance to learnable embeddings [14]. We first create a 2D grid of (x, y) coordinate pairs for each patch. The sinusoidal positional encoding can then be computed for each coordinate pair and dimension i ranging from 0 to $D/4 - 1$, resulting in a unique D-dimensional encoding for each patch:

$$\begin{aligned} E_{\text{sin-cos}}(x, y, 4i) &= \sin(x \cdot \omega_i) \\ E_{\text{sin-cos}}(x, y, 4i+1) &= \cos(x \cdot \omega_i) \\ E_{\text{sin-cos}}(x, y, 4i+2) &= \sin(y \cdot \omega_i) \\ E_{\text{sin-cos}}(x, y, 4i+3) &= \cos(y \cdot \omega_i) \\ \omega_i &= 1/10000^{4i/D} \end{aligned} \tag{3.6}$$

Note that for this encoding, the dimensionality D must be divisible by 4.

Global average-pooling. Instead of using a learnable classification token x_{class}, we use global average-pooling (GAP) to obtain the global representation of the image. Chen et al. [14] show that using GAP yields comparable performance to a classification token. Furthermore, the computational cost is reduced as GAP does not require an extra token to be processed through all attention layers, increasing the model efficiency. We calculate the global representation by averaging the activations across all patch embeddings from the final Transformer layer. This operation can be formulated as follows:

$$z_{\text{GAP}} = \frac{1}{N} \sum_{i=1}^{N} z_L^i \qquad z_L \in \mathbb{R}^{N \times D},\ z_{\text{GAP}} \in \mathbb{R}^D \tag{3.7}$$

where z_L^i represents the i-th token embedding from the output of the final Transformer layer L, and N the number of patch embeddings.

Our modified input to the transformer encoder can now be defined as:

$$z_0 = \text{SPT}(x) + E_{\text{sin-cos}} \qquad z_0 \in \mathbb{R}^{N \times D} \tag{3.8}$$

We combine Shifted Patch Tokenization with 2D sinusoidal positional encoding as input to our Transformer, and use global average-pooling to capture the final image representation.

3.2.2 Transformer encoder

The transformer encoder consists of L identical layers, and consists of two components: the multi-head self-attention mechanism (MHA) and a fully-connected Feed Forward Network (FFN) [75]. Layer normalization (LN) is applied before each component [5], and residual connections are used after each block [33]. This architecture can be formulated as follows:

$$z'_l = \text{MHA}(\text{LN}(z_{l-1})) + z_{l-1} \qquad l = 1 \dots L \tag{3.9}$$

$$z_l = \text{FFN}(\text{LN}(z'_l)) + z'_l \qquad l = 1 \dots L \tag{3.10}$$

Here, $z_l \in \mathbb{R}^{N \times D}$ is the output of layer l, and z_0 is the input to the first layer as defined in Eq. (3.8).

The FFN block uses two linear projections, and a GELU activation function [35]:

$$\text{FFN}(z) = \text{GELU}(z\mathbf{E}_1)\mathbf{E}_2 \qquad \mathbf{E}_1 \in \mathbb{R}^{D \times D_{hd}}, \mathbf{E}_2 \in \mathbb{R}^{D_{hd} \times D} \tag{3.11}$$

$$\text{GELU}(x) = x\Phi(x) \tag{3.12}$$

where $z \in \mathbb{R}^{N \times D}$ is the input embeddings, $\mathbf{E}_1$ and $\mathbf{E}_2$ the learnable projections, D_{hd} the hidden dimensionality of the FFN, and $\Phi(x)$ the standard Gaussian cumulative distribution function.

The self-attention mechanism works by first creating three matrices: the Query (Q), Key (K) and Value matrix (V). These matrices are obtained through a learnable linear projection ($\mathbf{E}^{QKV}$) of the input:

$$[Q, K, V] = z\mathbf{E}^{QKV} \qquad \mathbf{E}^{QKV} \in \mathbb{R}^{D \times 3D_h},\ Q, K, V \in \mathbb{R}^{N \times D_h} \tag{3.13}$$

where D_h corresponds to the dimension of each attention head. Next, the self-attention (SA) weights are computed as the scaled dot product between the query with all keys, followed by a softmax operation and multiplication with the Value matrix:

$$\text{SA}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{D_h}})V \tag{3.14}$$

The scaling factor $\sqrt{D_h}$ is used to prevent the softmax function from having small gradients. For multi-head self-attention (MSA), SA is extended to use multiple self-attention heads in parallel:

$$\begin{aligned} \text{MSA}(z) &= [\text{head}_1(z), \dots, \text{head}_h(z)]\mathbf{E}^O & \mathbf{E}^O &\in \mathbb{R}^{h \cdot D_h \times D} \\ \text{head}_i(z) &= \text{SA}(z\mathbf{E}_i^Q, z\mathbf{E}_i^K, z\mathbf{E}_i^V) & \mathbf{E}_i^Q, \mathbf{E}_i^K, \mathbf{E}_i^V &\in \mathbb{R}^{D \times D_h} \end{aligned} \tag{3.15}$$

Here, h is the number of attention heads and $\mathbf{E}_i^Q, \mathbf{E}_i^K$ and $\mathbf{E}_i^V$ are the learnable projections for the i-th head. The outputs of the individual heads is concatenated, and $\mathbf{E}^O$ is used to project the concatenated outputs back to the encoder dimensionality D. Moreover, D_h is usually set to D/h, keeping the computational cost of MSA similar to single-head attention with full dimensionality. Unfortunately, the computational complexity of self-attention scales quadratically $O(N^2)$ with the number of input tokens. And while various sub-quadratic approaches to self-attention have been proposed [20, 43], these approaches often come with different trade-offs in model performance. For this reason, we focus on standard self-attention in this thesis and leave further optimizations to future work.

Dropout. The original ViT architecture uses dropout [64] in the FFN and MHA components of their larger models [26]. Dropout is a regularization technique that randomly sets the activations of some neurons in a layer to 0 at each update during training time. This reduces co-adaption between neurons, which can greatly reduce the effects of overfitting. However, recent studies [7, 65] suggest that data augmentation is more effective than regularization techniques like dropout and stochastic depth [39], especially for smaller ViT models. Therefore, we exclude regularization such as dropout and stochastic depth for this thesis.

Layer Normalization. Layer Normalization (LN) is a technique introduced by Ba et al. [5] which helps stabilize training, allowing for faster convergence. Given an input vector $x \in \mathbb{R}^n$, Layer normalization works by re-centering and re-scaling the activations as follows:

$$\text{LN}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta \qquad \gamma, \beta \in \mathbb{R}^n \tag{3.16}$$

where γ and β are learnable gain and bias parameters, respectively. The mean μ and standard deviation σ are computed as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{3.17}$$

While layer normalization has been successfully applied to the ViT architecture [26], it also introduces computational overhead due to the extra statistics (μ, σ) that have to be calculated. To improve the efficiency of our architecture, we replace Layer Normalization with Root Mean Square Layer Normalization (RMSNorm) [82]. RMSNorm is an efficient alternative to Layer Normalization that removes the mean statistic, and has been successfully applied to Transformer LLMs like LLaMA [72]. The authors of RMSNorm argue that the re-centering of the activations contributes little to the success of layer normalization, and focus only on re-scaling the invariance and regularizing according to the root-mean-square (RMS) statistic:

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \cdot \gamma \tag{3.18}$$

$$\text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{3.19}$$

The resulting method is used as a drop-in replacement for all instances of layer normalization in our architecture, and further increases the efficiency of our model.

3.2.3 Prediction head

To get the final class predictions, we process the output of the transformer encoder through one final linear projection. This can be defined as follows:

$$y = z_{\text{GAP}}\mathbf{E}_{pred} \qquad z_{\text{GAP}} \in \mathbb{R}^D,\ \mathbf{E}_{pred} \in \mathbb{R}^{D \times K} \tag{3.20}$$

Here, z_{GAP} is the final image representation from the transformer encoder, as defined in Eq. (3.7), $\mathbf{E}_{pred}$ the learnable projection matrix, and K the number of output classes. The resulting logits $y \in \mathbb{R}^K$ are then passed through a softmax function to obtain the final class probabilities.

Deepfake classifier. When fine-tuning the model for deepfake detection, we have to replace the pre-trained prediction head with 1,000 output classes with a new classifier with 2 output classes (real or fake). As deepfake detection is a binary classification task, using a single linear layer would result in a significant loss of parameters. For this reason, we replace the prediction head with a simple classification network consisting of a small series of linear layers with ReLU [2] activations and optional dropout [64]:

$$x_0 = z_{\text{GAP}} \tag{3.21}$$
$$x_l = \text{Dropout}(\text{ReLU}(x_{l-1}\mathbf{W}_l)) \qquad l = 1 \dots L-1 \tag{3.22}$$
$$y = x_{L-1}\mathbf{W}_L \tag{3.23}$$
$$\text{ReLU}(x) = \max(0, x) \tag{3.24}$$

where L defines the amount of layers in the classification network. $\mathbf{W}_l \in \mathbb{R}^{D_{l-1} \times D_l}$ and $\mathbf{W}_L \in \mathbb{R}^{D_{L-1} \times D_L}$ are the learnable weight matrices, with $D_0 = D$ and $D_L = 2$. The dropout layers help prevent overfitting in the classifier network, and we gradually reduce the dimensionality of the data across layers ($D_l < D_{l-1}$) to progressively compress the information. This formulation gives us a flexible deepfake classifier that we can adjust to fit the needs of our ViT architecture.

3.3 BitLinear layer

In this section, we describe our implementation of the BitLinear layer as introduced by Wang et al. [76] for their BitNet LLM architecture. Specifically, we implement the BitLinear module proposed for BitNet b1.58 [56], which uses ternary weights $\{-1, 0, +1\}$ rather than the original binarized $\{-1, +1\}$ architecture. For our BitViT model, we directly replace all the linear layers in the transformer encoder of LinViT with BitLinear modules. The PyTorch [4] implementation of the BitLinear layer can be found on GitHub: `https://github.com/ingur/bitlinear-pytorch`.

The matrix multiplication in a standard full-precision linear layer can be expressed as follows:

$$y_i = (x\mathbf{W})_i = \sum_{j=1}^{n} x_j \mathbf{W}_{ij} \qquad x \in \mathbb{R}^{1 \times n}, \mathbf{W} \in \mathbb{R}^{n \times m}, y \in \mathbb{R}^{1 \times m} \tag{3.25}$$

where x is the input, $\mathbf{W}$ is the weight matrix and y the output. Using ternary weights constrained to the set $\{-1, 0, +1\}$, we can now replace the multiplication operation with pure addition operations. Let $\tilde{\mathbf{W}} \in \mathbb{R}^{n \times m}$ be the ternary weight matrix, where for each $\tilde{\mathbf{W}}_{ij} \in \{-1, 0, +1\}$. The multiplication operations can now be replaced by efficient addition or subtraction operations:

$$x_j \tilde{\mathbf{W}}_{ij} = \begin{cases} x_j, & \text{if } \tilde{\mathbf{W}}_{ij} = 1, \\ 0, & \text{if } \tilde{\mathbf{W}}_{ij} = 0, \\ -x_j, & \text{if } \tilde{\mathbf{W}}_{ij} = -1, \end{cases} \tag{3.26}$$

Which in turn allows us to express ternary matrix multiplication as:

$$\tilde{y}_i = \sum_{j=1}^{n} x_j \tilde{\mathbf{W}}_{ij} = \sum_{j:\tilde{\mathbf{W}}_{ij}=1}^{n} x_j - \sum_{j:\tilde{\mathbf{W}}_{ij}=-1}^{n} x_j \tag{3.27}$$

with $\tilde{y} \in \mathbb{R}^{1\times m}$ as the new output. Note that this also works for binary weights constrained to the set $\{-1, +1\}$, as the 0 weight only acts as a feature filter that introduces sparsity. The weight quantization uses an absmean quantization function, where we scale the weight matrix by its average absolute value and round each value to its nearest integer in the range $\{-1, 0, +1\}$:

$$\tilde{\mathbf{W}} = \text{RoundClip}(\frac{\mathbf{W}}{\beta + \epsilon}, -1, 1) \tag{3.28}$$

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))) \tag{3.29}$$

$$\beta = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} |W_{ij}| \tag{3.30}$$

where ϵ is a small floating point number to avoid division by zero errors. The activations are also quantized to 8-bit precision using absmax quantization [22] as defined in Eq. (3.31). Absmax quantization is a popular symmetric quantization technique which can also be used for post-training quantization [22]. Here, the activations are scaled into 8-bit range $[-128, 127]$ by multiplying the activations with 127 and dividing by the absolute maximum activation value:

$$\tilde{x} = \text{ActQuant}(x) = \text{RoundClip}(x \cdot \gamma, -128, 127) \tag{3.31}$$

$$\gamma = \frac{127}{\max(|x|, \epsilon)} \tag{3.32}$$

In full-precision linear layers, initialization methods such as Kaiming or Xavier initialization help maintain numerical stability throughout training [31, 32]. However, the non-linearity of the activation quantization does not preserve this variance. For this reason, RMSNorm (see Eq. (3.18)) is used before the activation quantization to help maintain numerical stability and preserve the variance. We ensure that RMSNorm is used before any linear layer throughout the model, including the patch tokenization process (SPT), before attention (MHA) and in the feedforward networks (FFN). Finally, the output activations are rescaled using β (Eq. (3.30)) and γ (Eq. (3.32)) to dequantize them to their original precision. The BitLinear layer can now be defined as follows:

$$y = \tilde{x}\tilde{\mathbf{W}} = \text{ActQuant}(\text{RMSNorm}(x))\tilde{\mathbf{W}} \times \frac{\beta}{\gamma} \tag{3.33}$$

This approach allows for future optimized hardware and kernel implementations to take advantage of the ternary weights for faster and more efficient matrix multiplication. However, for our training and experiment purposes, we apply the scaling factors before using the `F.Linear()` operation in PyTorch.

Straight-through estimator. Backpropagation requires all functions in our network to be differentiable. However, the round and clip functions used in the weight and activation quantization (see Eq. (3.29)) are non-differentiable. To address this issue, Wang et al. [76] use straight-through estimators (STEs) to allow the gradients to flow through these non-differentiable functions [6]. During training, we quantize the weights during each forward pass ($\tilde{\mathbf{W}}$). However, during the backward pass, the gradients are calculated with respect to the high-precision latent weights ($\mathbf{W}$).

3.4 Knowledge Distillation

Hinton et al. [37] introduced Knowledge Distillation as a training paradigm utilizing soft-labels from a strong teacher model to efficiently train smaller student models. Here, soft-labels refer to the raw logits as produced by the teacher network, as opposed to the maximum of scores, which gives a hard-label. Moreover, knowledge distillation can also be used to compress the knowledge of larger models into smaller models. Touvron et al. [73] expanded upon their work by introducing a new distillation strategy designed for ViT models, resulting in the Data-Efficient Image Transformer (DeiT). Using this strategy on ImageNet-1k, DeiT is able to outperform the original ViT-B model pretrained on the JFT-300M dataset [66].

Based on the results of Touvron et al. [73], we implement hard-label distillation due to its increased performance over soft-label distillation. Let y be the ground-truth label, and $y_t = \text{argmax}_c z_t(c)$ the one-hot encoded hard decision of the teacher model. We append a learnable distillation token to the initial patch embeddings z_0, which interacts with the other embeddings through self-attention, similarly to a class token x_{class}. Let z_s represent the final output of our Vision Transformer, and z_d the final output distillation token. The hard-label distillation loss can now be defined as:

$$\mathcal{L}^{\text{hard}}_{\text{global}} = (1-\alpha)\mathcal{L}_{\text{CE}}(\psi(z_s), y) + \alpha\mathcal{L}_{\text{CE}}(\psi(z_d), y_t) \tag{3.34}$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy function, and ψ the softmax function. Additionally, α controls the tradeoff between main loss and distillation loss, and is set to 0.5 by default. This approach enables our ViT model to efficiently learn from a strong teacher classifier model, further improving its performance while being more data-efficient.

3.5 Post-training Quantization

To be able to compare the effectiveness of quantization-aware training in BitViT against post-training quantization methods, we integrate two commonly used PTQ techniques: absolute max quantization and zero-point quantization.

To quantize the weights or activations x in our linear layers in LinViT after training, we can extend the absmax quantization formulated in Eq. (3.31) for BitLinear to any range $[-Q_b, Q_b]$:

$$\tilde{x} = \text{AbsMaxQuant}(x) = \text{RoundClip}(x \cdot \gamma, -Q_b, Q_b) \tag{3.35}$$

$$\gamma = \frac{Q_b}{\max(|x|, \epsilon)} \tag{3.36}$$

where $Q_b = 2^{b-1}$ and ϵ is a small floating point number to avoid division by zero errors. This function works well for symmetric input distributions, where the positive and negative values are evenly distributed around zero. However, the effectiveness of this method is limited when quantizing asymmetric distributions not centered around 0. To reduce the quantization error for asymmetric distributions, we can use zero-point quantization:

$$\tilde{x} = \text{ZeroPointQuant}(x) = \text{RoundClip}(x \cdot \gamma + z, -Q_b, Q_b) \tag{3.37}$$

$$\gamma = \frac{2 \cdot Q_b}{\max(x) - \min(x)} \tag{3.38}$$

$$z = \text{round}(-\min(x) \cdot \gamma - Q_b) \tag{3.39}$$

Here, the scaling factor γ is now calculated using the full range of the input, rather than using the absolute max value, and z represents the zero-point which allows the quantization range to shift based on the input distribution.

rewrite this While this method reduces the quantization error on asymmetric distributions, outliers can still reduce the quantization precision of all other values [22]. Furthermore, as discussed in Sec. 2.4, sub 8-bit quantization suffers from performance degradation due to the significant difference between the original and quantized weights. However, due to time and resource constraints, we will not implement advanced PTQ methods such as GPTQ [28], QuIP [11], RepQuant [51] and SmoothQuant+ [59] for our experiments.

4 | Experiments

In this chapter, we discuss the different experiments on our LinViT and BitViT architecture and the corresponding results. First, we pre-train and compare the performance of LinViT and BitViT models on the ImageNet-1k dataset [62] in Sec. 4.1. In Sec. 4.2, we analyze the effects of post-training quantization on LinViT, and compare it to the effectiveness of quantization-aware training in BitViT. Finally, we fine-tune both models for deepfake detection on the FaceForensics++ [61] and Celeb-DF [50] datasets in Sec. 4.3, and analyze the results.

4.1 Pre-training

To properly assess the image classification capabilities of our ViT architecture, we pre-train the LinViT and BitViT models on the ImageNet-1k dataset, which is prepared as described in Sec. 3.1. Additionally, all our experiments throughout this chapter are implemented in PyTorch [4] and performed on a single NVIDIA A100-SXM4 GPU with 40GB of VRAM memory. The full pre-training settings we use are summarized in Tab. A.1.

Number of epochs. Due to computational and time constraints, we train our models for 50 epochs, as opposed to the 90 epochs used by Beyer et al. [7] for their baseline and 300 epochs commonly used for Vision Transformers [26, 73]. While the amount of training epochs has a great impact on the final performance of the models, we can still use these early checkpoints to perform a fair comparison between our architectures given the computational constraints.

Batch size. Based on Beyer et al. [7] and considering the size of our dataset, we use a batch size of 1024 for pre-training on ImageNet-1k. To account for the limited amount of available GPU memory available, gradient accumulation is used to emulate the target batch size for our training. For the LinViT model, 2 accumulation steps were used on a batch size of 512, whereas the BitViT model required 4 accumulation steps with a batch size of 256 due to the increased memory overhead of quantization-aware training. This keeps our effective batch size 1024 for both models.

Model parameters. Table 4.1 summarizes the model configuration used throughout our experiments. We chose to base our model parameters on the ViT-S/16 model configuration introduced for DeiT, as it provides a good tradeoff between common training hardware compatibility, throughput and accuracy [73]. Both the LinViT and BitViT models are trained using these base parameters.

Following Dosovitskiy et al. [26], the hidden dimensionality D_{hd} of the FFN block is set to 4 times the embedding dimensionality of the transformer encoder D: $4\times384 = 1536$. Additionally, the dimensions per attention head are set to $D_h = D/\#\text{heads} = 64$.

Training parameters. The models are trained using the Adam optimizer with decoupled weight decay (AdamW) [55]. Based on the results of Steiner et al. [65], we set the initial

#Layers	#Heads	Embedding dim	FFN dim	Patch size	Image size	#Params
12	6	$D = 384$	$D_{hd} = 1536$	16×16	224×224	22M

Table 4.1: Base model parameters for LinViT and BitViT, based on the VIT-S/16 configuration as introduced by Touvron et al. [73].

learning rate to 0.001, $\beta_1 = 0.9, \beta_2 = 0.999$ and a low weight decay of $1e-4$. Furthermore, to schedule our learning rate (LR), Cosine Annealing LR decay [54] is used with $T_{max} = 90$ and a linear warmup of 5 epochs.

For BitNet, Wang et al. [76] explored using higher peak learning rates to accelerate the optimization process. This was done because small updates on the latent weights of BitLinear often make no difference on the quantized weights, which makes BitLinear models more stable than their full-precision counterparts. However, while increasing the peak learning rate in our experiments maintained training stability, we did not see an increased performance within our 50 epoch training schedule.

Furthermore, when training full-precision models, weight decay acts as a regularizer, constraining the magnitude of large weights. This method helps prevent overfitting and increases training stability [52]. For BitLinear-based models, however, weight decay is applied to the latent weights, rather than the ternary weights. In this case, the magnitude of the latent weights can be interpreted as a confidence score [56]. Therefore, high weight decay values lead to lower confidence scores for the ternary weights, causing them to change more frequently. In our testing, this led to decreased performance in training (see Fig. A.1). For this reason, we opted to use the lower weight decay in our experiments, rather than the values used for DeiT.

Knowledge distillation. As stated in Sec. 3.4, hard-label distillation loss is used due to its increased performance over soft-label distillation [73]. We use the default $\alpha = 0.5$, which gives equal importance to the true labels and the hard labels from the teacher. For the teacher model, we use the pre-trained EfficientNetV2-S model [69] from PyTorch. This fast CNN model achieves a top-1 accuracy of 84.228% on ImageNet-1k, making it a suitable teacher model for our experiments.

Touvron et al. [73] also reported that increasing the number of epochs significantly improves the performance of the model with distillation. This also means that while we already observe slight accuracy improvements using distillation for only 50 epochs, we can also expect even greater improvements with longer training.

Data augmentation. As described in Sec. 3.1, we experiment with a combination of recent data augmentation techniques: RandAugment [19], MixUp [83] and CutMix [81]. During training, we first randomly crop the input images to 224x224 pixels, followed by a random horizontal flip. Note that for validation, we use a center crop instead to ensure consistent evaluation. Following the results of Beyer et al. [7], we apply RandAugment with 2 operations and set the magnitude to 10. We then normalize the images using $\mu = [0.485, 0.456, 0.406]$ and $\sigma = [0.229, 0.224, 0.225]$, based on the ImageNet-1k dataset. For Mixup and Cutmix, we experimented with combining both augmentations using $\alpha_{\text{mixup}} = 0.8$ and $\alpha_{\text{cutmix}} = 1.0$, with probability $p = [0.5, 0.5]$. However, we found that using only MixUp with $\alpha_{\text{mixup}} = 0.2$ performed slightly better (see Fig. A.2). This aligns with the results of Steiner et al. [65], who found that high levels of data augmentation hurt the performance of smaller models.

Automatic Mixed Precision Training. To further accelerate training, we enable Automatic Mixed Precision (AMP) training as implemented in PyTorch [4]. This allows us to use

cheaper 16-bit floating-point precision in our matrix multiplication operations, which reduces the training time and memory footprint of our model at no loss of accuracy in our experiments.

4.1.1 Evaluation

To measure the performance of our trained models, we focus on two metrics: top-1 accuracy on the ImageNet-1k validation set and model size. Figure 4.1 shows the validation loss and top-1 accuracy of BitViT and LinViT during the 50 epoch training schedule.

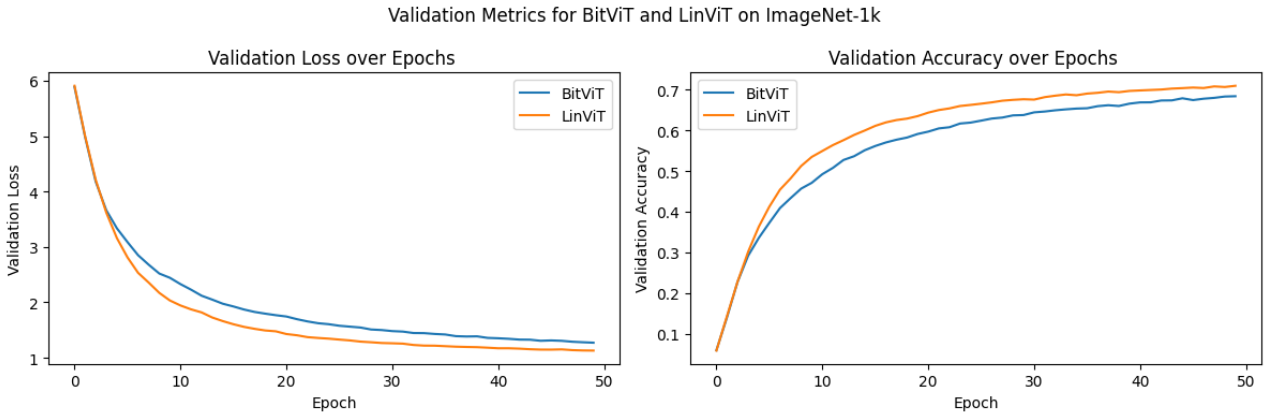

Figure 4.1: Comparison of validation loss and top-1 accuracy for BitViT and LinViT models during pre-training on ImageNet-1k.

4.2 Quantization

In order to analyze the effects of post-training quantization on our pre-trained LinViT model, we implement the absmax and zeropoint quantization methods as defined in Eqs. (3.35) and (3.37). We replace all the existing linear layers with custom layers that allow us to configure the desired quantization technique and bit-precision for weight and activation quantization. Next, we copy the pre-trained weight data to the new layers as latent weights, and test the quantized model performance on the ImageNet-1k validation set.

4.2.1 Evaluation

4.3 Fine-tuning

In this section, we investigate the adaptability of our pre-trained LinViT and BitViT models to deepfake detection tasks. For these experiments, we use the subsets of the FaceForensics++ and Celeb-DF-V2 datasets as prepared in Sec. 3.1. Furthermore, we also fine-tune the pre-trained EfficientNetV2-S model on the same datasets to serve as a comparison baseline.

Training settings. For fine-tuning, we retain most of the settings used in pre-training. However, we only train for 10 epochs and reduce the batch size to 256 to account for the reduced dataset size. Moreover, we disable knowledge distillation to assess the raw adaptability to the binary classification task, and use cross-entropy loss. The RandAugment, CutMix and MixUp data augmentations are also disabled, as they would significantly distort the small details needed to correctly detect deepfakes. The full fine-tune settings are summarized in Tab. A.2.

Layer freezing. We implement a layer freezing strategy to preserve the learned representations from pre-training. Specifically, we freeze all the model weights and only fine-tune the last N

Model	PTQ	(Q_w, Q_x)	Top-1 val acc	Model size
LinViT	-	(32, 32)	0.7094	88.15 MB
BitViT	-	(2, 8)	0.6840	6.08 MB
LinViT	absmax	(8, 8)	0.7074	22.04 MB
LinViT	absmax	(6, 8)	0.7072	16.53 MB
LinViT	absmax	(4, 8)	0.6845	11.02 MB
LinViT	absmax	(3, 8)	0.4169	8.26 MB
LinViT	absmax	(2, 8)	0.0010	5.51 MB
LinViT	zeropoint	(8, 8)	0.7070	22.04 MB
LinViT	zeropoint	(6, 8)	0.7004	16.53 MB
LinViT	zeropoint	(4, 8)	0.3569	11.02 MB
LinViT	zeropoint	(3, 8)	0.0011	8.26 MB
LinViT	zeropoint	(2, 8)	0.0010	5.51 MB

Table 4.2: Comparison of top-1 validation accuracy and model size for BitViT and LinViT using various levels of absmax and zeropoint quantization (see Eqs. (3.35) and (3.37)), where (Q_w, Q_x) denotes the bit-precision of the weights and activations, respectively.

transformer blocks. For our experiments, we use $N = 6$ to balance between retaining the pre-learned features and adapting to the deepfake data.

Prediction head. To adapt LinViT for deepfake detection, we replace the existing prediction head with our custom classification network as described in Sec. 3.2.3. For our hidden layer output sizes, we use $[192, 96, 48, 24]$ to gradually compress the feature space while still reducing the parameter space compared to the original classification head. For dropout, we use a low value of 0.2 to help prevent overfitting and increase generalization of our fine-tuned model.

4.3.1 Evaluation

Model	Dataset	Test acc
EfficientNetV2-S	FF++	0.9659
EfficientNetV2-S	C-DF	0.9262
LinViT	FF++	0.9629
LinViT	C-DF	0.9349
BitViT	FF++	0.9580
BitViT	C-DF	0.9244

Table 4.3: test accuracy for deepfake detection on FaceForensics++ (FF++) and Celeb-DF-V2 (C-DF). Comparison between the EfficientNetV2-S CNN, LinViT and BitViT.

5 | Conclusion

Bibliography

[1] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, December 2018. doi: 10.1109/wifs.2018.8630761.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019. URL `https://arxiv.org/abs/1803.08375`.

[3] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary Neural Networks for Resource-Efficient AI Applications. *arXiv:1609.00222*, 2017.

[4] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL `https://arxiv.org/abs/1607.06450`.

[6] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR*, abs/1308.3432, 2013. URL `http://arxiv.org/abs/1308.3432`.

[7] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain ViT baselines for ImageNet-1k. *arXiv:2205.01580*, 2022.

[8] Omkar Bhilare, Rahul Singh, Vedant Paranjape, Sravan Chittupalli, Shraddha Suratkar, and Faruk Kazi. Deepfake cli: Accelerated deepfake detection using fpgas, 2022. URL `https://arxiv.org/abs/2210.14743`.

[9] Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age, 2018. URL `https://arxiv.org/abs/1710.08092`.

[10] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021. URL https://arxiv.org/abs/2104.14294.

[11] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. Quip: 2-bit quantization of large language models with guarantees, 2024. URL https://arxiv.org/abs/2307.13304.

[12] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention approximation, 2021. URL https://arxiv.org/abs/2110.15343.

[13] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification, 2021.

[14] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers, 2021. URL https://arxiv.org/abs/2104.02057.

[15] Bobby Chesney and Danielle Citron. Deep fakes: A looming challenge for privacy. *Calif. L. Rev.. California Law Review*, 107(IR):1753, 2019. URL http://lawcat.berkeley.edu/record/1136469.

[16] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. URL https://arxiv.org/abs/1610.02357.

[17] Davide Alessandro Coccomini, Nicola Messina, Claudio Gennaro, and Fabrizio Falchi. *Combining EfficientNet and Vision Transformers for Video Deepfake Detection*, page 219–229. Springer International Publishing, 2022. ISBN 9783031064333. URL http://dx.doi.org/10.1007/978-3-031-06433-3_19.

[18] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830*, 2016.

[19] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019. URL https://arxiv.org/abs/1909.13719.

[20] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

[21] Sowmen Das, Selim Seferbekov, Arup Datta, Md. Saiful Islam, and Md. Ruhul Amin. Towards solving the deepfake problem : An analysis on improving deepfake detection using dynamic face augmentation, 2021. URL https://arxiv.org/abs/2102.09603.

[22] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *arXiv:2208.07339*, 2022.

[23] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017. URL https://arxiv.org/abs/1708.04552.

[24] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) preview dataset, 2019. URL https://arxiv.org/abs/1910.08854.

[25] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset, 2020. URL https://arxiv.org/abs/2006.07397.

[26] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929*, 2020.

[27] Dayou Du, Gu Gong, and Xiaowen Chu. Model quantization and hardware acceleration for vision transformers: A comprehensive survey, 2024. URL https://arxiv.org/abs/2405.00314.

[28] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

[29] N. Fufour, P. Gully, A. Karlsson, A.V. vorbyov, T. Leung, J. Childs, and C. Bregler. Deepfakes detection dataset by google & jigsaw, 2019.

[30] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021. URL https://arxiv.org/abs/2103.13630.

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[34] Yefei He, Zhenyu Lou, Luoming Zhang, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Bivit: Extremely compressed binary vision transformer, 2023. URL https://arxiv.org/abs/2211.07091.

[35] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

[36] Young-Jin Heo, Young-Ju Choi, Young-Woon Lee, and Byung-Gyu Kim. Deepfake detection scheme based on vision transformer and distillation, 2021. URL https://arxiv.org/abs/2104.01353.

[37] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[38] Tao Hu, Honggang Qi, Qingming Huang, and Yan Lu. See better before looking closer: Weakly supervised data augmentation network for fine-grained visual classification, 2019. URL https://arxiv.org/abs/1901.09891.

[39] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016. URL https://arxiv.org/abs/1603.09382.

[40] Mingqiang Huang, Junyi Luo, Chenchen Ding, Zikun Wei, Sixiao Huang, and Hao Yu. An integer-only and group-vector systolic accelerator for efficiently mapping vision transformer on edge. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(12):5289–5301, 2023. doi: 10.1109/TCSI.2023.3312775.

[41] Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. Billm: Pushing the limit of post-training quantization for llms, 2024. URL https://arxiv.org/abs/2402.04291.

[42] Liming Jiang, Ren Li, Wayne Wu, Chen Qian, and Chen Change Loy. Deeperforensics-1.0: A large-scale dataset for real-world face forgery detection, 2020. URL https://arxiv.org/abs/2001.03024.

[43] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

[44] Pavel Korshunov and Sebastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection, 2018. URL https://arxiv.org/abs/1812.08685.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[46] Phuoc-Hoan Charles Le and Xinlin Li. Binaryvit: Pushing binary vision transformers towards convolutional models, 2023. URL https://arxiv.org/abs/2306.16678.

[47] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets. *arXiv:2112.13492*, 2021.

[48] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.21.

[49] Yuezun Li and Siwei Lyu. Exposing deepfake videos by detecting face warping artifacts, 2019.

[50] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A new dataset for deepfake forensics. *CoRR*, abs/1909.12962, 2019. URL http://arxiv.org/abs/1909.12962.

[51] Zhikai Li, Xuewen Liu, Jing Zhang, and Qingyi Gu. Repquant: Towards accurate post-training quantization of large transformer models via scale reparameterization, 2024. URL https://arxiv.org/abs/2402.05628.

[52] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization?, 2021. URL https://arxiv.org/abs/2106.11309.

[53] Zhenhua Liu, Yunhe Wang, Kai Han, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer, 2021. URL https://arxiv.org/abs/2106.14156.

[54] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983.

[55] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[56] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. *arXiv:2402.17764*, 2024.

[57] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. ISSN 0893-6080. URL https://www.sciencedirect.com/science/article/pii/S0893608097000117.

[58] Huy H. Nguyen, Junichi Yamagishi, and Isao Echizen. Exploring self-supervised vision transformers for deepfake detection: A comparative analysis, 2024. URL https://arxiv.org/abs/2405.00355.

[59] Jiayi Pan, Chengcan Wang, Kaifu Zheng, Yangguang Li, Zhenyu Wang, and Bin Feng. Smoothquant+: Accurate and efficient 4-bit post-training weightquantization for llm, 2023. URL https://arxiv.org/abs/2312.03788.

[60] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners, 2019.

[61] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. *CoRR*, abs/1901.08971, 2019. URL http://arxiv.org/abs/1901.08971.

[62] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015. URL https://arxiv.org/abs/1409.0575.

[63] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. URL https://arxiv.org/abs/1409.1556.

[64] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[65] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers, 2022. URL https://arxiv.org/abs/2106.10270.

[66] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era, 2017. URL https://arxiv.org/abs/1707.02968.

[67] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[68] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. URL https://arxiv.org/abs/1905.11946.

[69] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training, 2021. URL https://arxiv.org/abs/2104.00298.

[70] Vrizlynn L. L. Thing. Deepfake detection with deep learning: Convolutional neural networks versus transformers, 2023. URL https://arxiv.org/abs/2304.03698.

[71] Ruben Tolosana, Sergio Romero-Tapiador, Julian Fierrez, and Ruben Vera-Rodriguez. Deepfakes evolution: Analysis of facial regions and fake detection performance, 2020. URL https://arxiv.org/abs/2004.07532.

[72] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and Efficient Foundation Language Models. *arXiv:2302.13971*, 2023.

[73] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2021.

[74] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers, 2021. URL https://arxiv.org/abs/2103.17239.

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Conference on Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[76] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: Scaling 1-bit Transformers for Large Language Models. *arXiv:2310.11453*, 2023.

[77] Run Wang, Felix Juefei-Xu, Lei Ma, Xiaofei Xie, Yihao Huang, Jian Wang, and Yang Liu. Fakespotter: A simple yet robust baseline for spotting ai-synthesized fake faces, 2020. URL https://arxiv.org/abs/1909.06122.

[78] Deressa Wodajo and Solomon Atnafu. Deepfake video detection using convolutional vision transformer, 2021. URL https://arxiv.org/abs/2102.11126.

[79] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. URL https://arxiv.org/abs/2211.10438.

[80] Xin Yang, Yuezun Li, and Siwei Lyu. Exposing deep fakes using inconsistent head poses, 2018. URL https://arxiv.org/abs/1811.00661.

[81] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019. URL https://arxiv.org/abs/1905.04899.

[82] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019. URL https://arxiv.org/abs/1910.07467.

[83] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018. URL https://arxiv.org/abs/1710.09412.

[84] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, October 2016. ISSN 1558-2361. URL http://dx.doi.org/10.1109/LSP.2016.2603342.

[85] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning, 2021. URL https://arxiv.org/abs/2104.08500.

[86] Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng Zhou, and Jason K. Eshraghian. Scalable matmul-free language modeling, 2024. URL https://arxiv.org/abs/2406.02528.

[87] Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason K. Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks, 2024. URL https://arxiv.org/abs/2302.13939.

A | Appendix

A.1 Pre-training

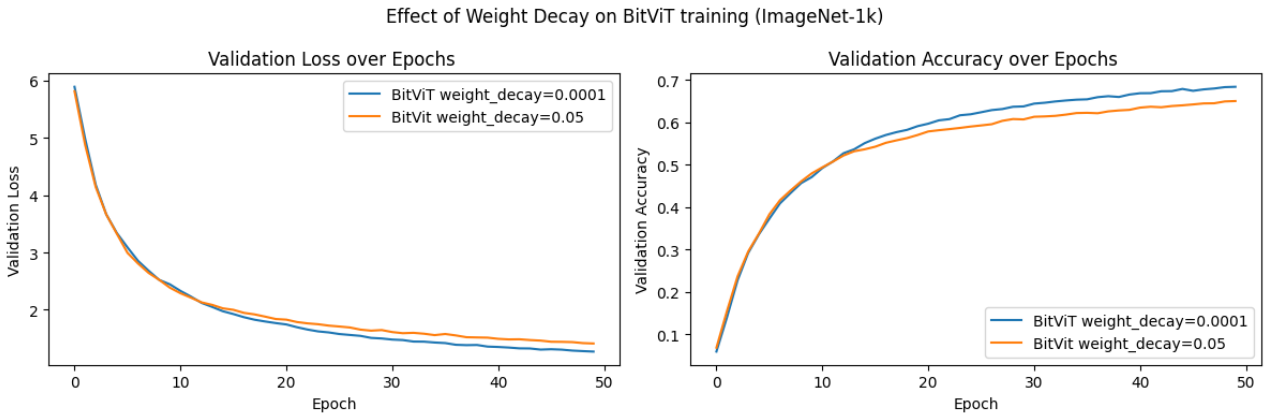

Figure A.1: Enter Caption

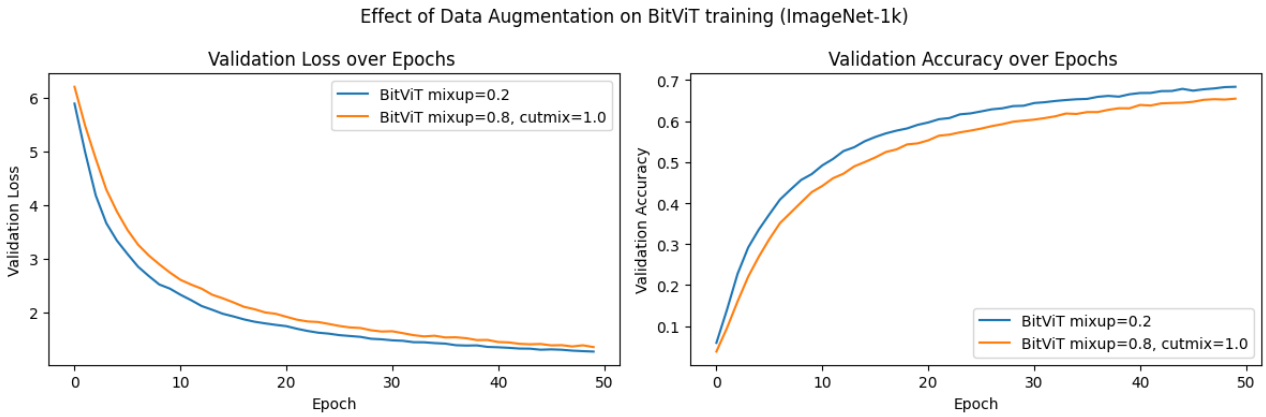

Figure A.2: Enter Caption

Hyperparameter	Value
Epochs	50

Table A.1: Caption

A.2 Quantization

A.3 Fine-tuning

Hyperparameter	Value
Epochs	10

Table A.2: Caption