Bytes Are All You Need: Transformers Operating Directly On File Bytes

Maxwell Horton, Sachin Mehta, Ali Farhadi, Mohammad Rastegari
Apple

Abstract

Modern deep learning approaches usually transform inputs into a modality-specific form. For example, the most common deep learning approach to image classification involves decoding image file bytes into an RGB tensor which is passed into a neural network. Instead, we investigate performing classification directly on file bytes, without the need for decoding files at inference time. Using file bytes as model inputs enables the development of models which can operate on multiple input modalities. Our model, ByteFormer*, achieves an ImageNet Top-1 classification accuracy of* 77.33% *when training and testing directly on TIFF file bytes using a transformer backbone with configuration similar to DeiT-Ti (*72.2% *accuracy when operating on RGB images). Without modifications or hyperparameter tuning, ByteFormer achieves* 95.42% *classification accuracy when operating on WAV files from the Speech Commands v2 dataset (compared to state-of-the-art accuracy of* 98.7%*). Additionally, we demonstrate that ByteFormer has applications in privacy-preserving inference. ByteFormer is capable of performing inference on particular obfuscated input representations with no loss of accuracy. We also demonstrate ByteFormer's ability to perform inference with a hypothetical privacy-preserving camera which avoids forming full images by consistently masking* 90% *of pixel channels, while still achieving* 71.35% *accuracy on ImageNet. Our code will be made available at* `https://github.com/apple/ml-cvnets/tree/main/examples/byteformer`.

1. Introduction

Deep learning inference usually involves explicit modeling of the input modality. For example, Vision Transformers (ViTs) [7] explicitly model the 2D spatial structure of images by encoding image patches into vectors. Similarly, audio inference often involves computing spectral features (such as MFCCs [25]) to pass into a network [10, 18]. When a user wants to perform inference on a file stored on disk (e.g. a JPEG image file or an MP3 audio file), the user must first decode the file into a modality-specific representation (e.g. an RGB tensor or MFCCs), as in Figure 1a.

The practice of decoding inputs into a modality-specific representation has two main drawbacks. First, it requires hand-crafting an input representation and a model stem for each input modality. Recent works such as PerceiverIO [14] and UnifiedIO [24] have shown that Transformer backbones can be used for a variety of different tasks. However, these methods still require modality-specific input preprocessing. For instance, PerceiverIO decodes image files into $[H \times W, C]$ tensors before passing them into the network. Other modalities input to PerceiverIO are processed into different forms. We hypothesize that it's possible to remove all modality-specific input preprocessing by performing inference directly on file bytes.

The second drawback of decoding inputs into a modality-specific representation is that it reduces privacy by exposing the data being analyzed. Consider the case of a smart-home device that performs inference on RGB images. If an adversary accesses this model input, the user's privacy might be compromised. We hypothesize that inference can instead be performed on privacy-preserving inputs.

To address these drawbacks, we note that a common property of many input modalities is that they can be stored as file bytes. Thus, we use file bytes (without any decoding) as inputs to our model at inference time (Figure 1b). We use a modified Transformer [39] architecture for our model, given their ability to handle a variety of modalities [14, 24] and variable-length inputs. We call our model ByteFormer.

We demonstrate the efficacy of ByteFormer on ImageNet [6] classification, achieving 77.33% accuracy on files stored in the TIFF format. Our model uses transformer backbone hyperparameters chosen in DeiT-Ti [38] (which achieves 72.2% accuracy on RGB inputs). We also demonstrate strong results on PNG and JPEG files. Additionally, we demonstrate that our classification model can achieve 95.8% accuracy on Speech Commands v2 [42], comparable to state-of-the-art (98.7%) [18], *without any architecture changes or hyperparameter tuning*.

Because ByteFormer can handle a variety of input representations, we can also use it to operate on privacy-preserving inputs. We demonstrate that we can remap in-

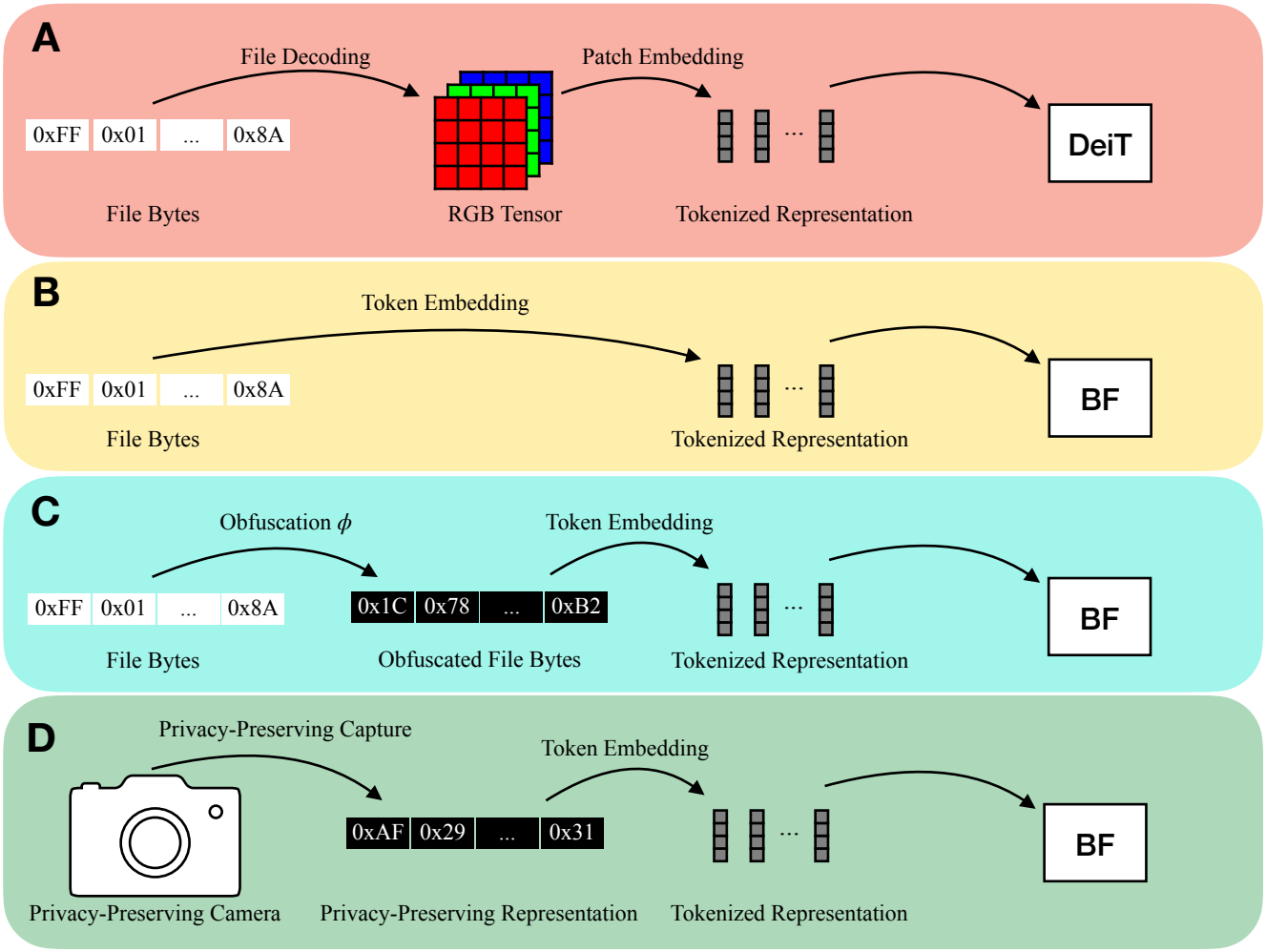

Figure 1. An overview of our ByteFormer (BF) compared to standard inference with DeiT [38]. (A): File bytes are read from disk and converted to an RGB tensor using a standard image decoder. A patch embedding creates tokens from the RGB representation. (B): File bytes on disk are directly used as tokens, and projected into learned embeddings. (C): Similar to (B), but we apply an obfuscation function ϕ. (D): We capture a privacy-preserving representation with a custom camera, and perform token embedding from this representation.

put byte values using a permutation function $\phi : [0, 255] \rightarrow [0, 255]$ (Figure 1c) to obfuscate inputs without losing accuracy. Although this does not guarantee cryptography-level security, we demonstrate how this method can be used as a building block for obfuscating inputs to a learning system.

Stronger privacy can be obtained by performing inference with ByteFormer on a partially-formed image (Figure 1d). We demonstrate that ByteFormer is capable of training on images with 90% of the pixels masked while still achieving 71.35% accuracy on ImageNet [6]. ByteFormer does not require information about the specific location of unmasked pixels. Our representation passed to our model maintains privacy by avoiding a standard image capture.

In summary, our contributions are: (1) we develop ByteFormer, a model capable of performing inference on file bytes. (2) We show that ByteFormer achieves strong performance on a variety of image and audio file encodings, without the need for architectural changes or hyperparameter tuning. (3) We demonstrate application of ByteFormer to privacy-preserving inputs. (4) We analyze properties of ByteFormers trained to classify images and audio directly from file bytes. (5) We will release our code at `https://github.com/apple/ml-cvnets/tree/main/examples/byteformer`.

2. Related Work

Architectures With Multimodal Inputs: A few methods have explored the idea of feeding different input modalities into the same network for processing. Perceiver IO [14] demonstrates that a Transformer [39] architecture with cross-attention input can be used for a variety of different tasks. Their method differs from ours because their inputs

are processed with modality-specific preprocessing. For example, images are loaded into a $[H \times W, C]$ buffer, which differs from the model's treatment of text. By contrast, our method can classify images directly from file bytes. To our knowledge, we are the first to explore models that directly consume file bytes without modality-specific processing.

Other recent works that model multiple modalities with a single model or a single embedding space (but still require input-specific processing) include Unified IO [24], CLIP [33], and LQAE [21].

Alternate Image Input Representations: Previous works have explored using alternate input representations for images. In [11], the authors perform partial JPEG decoding, stopping when Discrete Cosine Transform [26] coefficients are formed. They modify CNNs [12] to ingest this new representation. In [31], a similar method is used with Transformers. Our work differs in that we perform no decoding of file bytes at inference time.

Privacy-Preserving Inference: We demonstrate applications of our model to privacy-preserving inference. A few works have examined secure inference in Multi-Party Communication (MPC) settings [41, 19, 35, 37, 16]. The focus of these works is to perform inference securely on a remote machine using decrypted data on a local machine. We differ from these methods in that our privacy-preserving systems add a layer of privacy to the data on a local machine, and inference is performed locally. Thus, our approach complementary to theirs.

Compressive Sensing: Our privacy-preserving camera is inspired by works in compressive sensing [8]. Related works use image input masking with a single-pixel camera to capture an image over multiple exposures with different masks [9, 13]. Instead, we experiment with a single masked capture on a multi-pixel camera.

3. Overview of Common File Encodings

When performing inference with a standard model, the choice of file encoding is irrelevant. For example, it doesn't matter whether an image is stored as a JPEG or PNG file if it will be decoded into an RGB tensor. By contrast, ByteFormer performs inference on file bytes. In this case, the choice of file encoding matters. This section provides an overview of common file encodings for images (subsection 3.1) and audio (subsection 3.2). File encoding methods typically contain a large number of optional settings that influence the resulting file bytes. We use default settings provided by `PIL` [4] or `scipy` [40] software packages unless otherwise stated.

3.1. Image File Encodings

fHWC: We use fHWC as an abbreviation for "flattened tensors in height, width, channel order." It refers to uint8 image bytes stored in HWC order without any file headers. It is not common to store images in this way, since they cannot be decoded without pre-existing knowledge of their height and width. This serves as a strong baseline.

fCHW: This format is similar to fHWC, but images are stored in "CHW" order.

TIFF: The TIFF file encoding [32] allows for many custom configurations. For our experimentation, we use the default settings provided by PIL. This results in a format similar to fHWC, but with the addition of TIFF image headers describing configuration options and the image size.

PNG: The PNG format [2] contains headers describing PNG configuration options, followed by rows of image data stored in "IDAT" chunks. Each IDAT chunk contains a byte describing the filtering method used for that row's data. The filtering method applies an offset to the row's data based on neighboring pixel values. Thus, our PNG file contains rows of RGB data, with offsets applied, interrupted by occasional bytes that contain file encoding settings. We do not use the optional `zlib` compression that PNG allows.

JPEG: JPEG [43] encodes images by applying a series of transformations to compress the image before serialization. The RGB image is converted to YCbCr, then downsampled in the chroma channels, then passed through a Discrete Cosine Transform [26], then quantized using coefficients determined by the JPEG quality factor. The quality factor determines the level of compression, with 100 denoting no compression due to quantization, and lower values indicating stronger compression. After quantization, the coefficients are encoded via a run-length encoding, followed by a Huffman encoding [34]. Note that Huffman codes are not byte-aligned, e.g. they can cross byte boundaries. We expect this to make our modeling task more difficult.

3.2. Audio File Encodings

WAV: The WAV file encoding [17] stores audio signals represented as a sequence of amplitudes. We use single-channel (mono) audio files. The most common configuration options are the bit depth and the frequency. The bit depth corresponds to the precision with which amplitude values are stored. We experiment with a variety of bit depths, storing audio with 8-bit unsigned integers, 16-bit integers, 32-bit integers, and 32-bit floating-point values. The frequency corresponds to how often amplitude values are chosen. We use 16 kHz, a standard choice for audio [42].

MP3: MP3 [30] uses a perceptual compression method that removes portions of audio that are difficult for humans to detect. The remaining portions are recorded in frequency space. An MP3 file contains a series of frames. Each frame contains a header with encoding settings, followed by the encoded signal in frequency space. We use standard settings for MP3 provided by the `pydub` [36] software package. We expect MP3 encodings to be more difficult to handle than WAV files due to the compression applied.

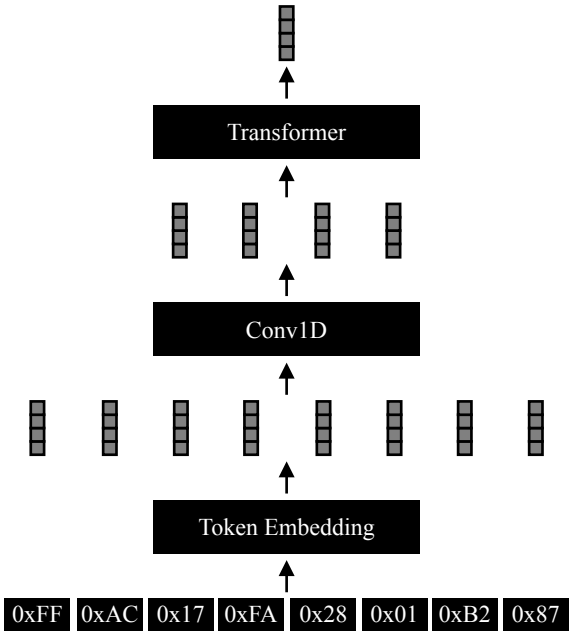

Figure 2. An overview of ByteFormer. We map byte values to learned vectors using a learned token embedding. Next, we apply a Conv1D to reduce the token dimension. Finally, we apply a transformer with shifted window attention and downsampling.

4. Methods

First, we discuss our method for performing inference on file bytes (subsection 4.1). Then, we discuss how to use our method with image obfuscation techniques to enable privacy-preserving inference (subsection 4.2). Finally, we discuss how to use our method with a privacy-preserving camera to perform inference without constructing full images (subsection 4.3).

4.1. Inference on File Bytes

4.1.1 Preprocessing

Some of our file encodings such as TIFF are not frequently used in machine learning datasets. To allow for comparisons on a single dataset across a variety of file encodings, we must re-encode files with different file encodings.

At training time, we decode the file (e.g. read the contents into an RGB tensor in the case of images), then perform standard training augmentation (e.g. random cropping in the case of images), then save the result in the desired file encoding. We find that standard training augmentation is important for model accuracy. Thus, our *training* method is implicitly dependent on the input modality due to our augmentation.

At *inference* time, we do not need knowledge of the input modality. We only need to ensure that our model inputs use the correct file encoding. For example, for TIFF experiments on ImageNet, we precompute 224×224 crops of the validation images and save them in the TIFF format. Such preprocessing is only necessary because the ImageNet validation set is not already stored in the desired format.

Model	**Data format**	$\mathbb{E}[S]$	$\mathbb{E}[L_t]$	**Top-1**
DeiT-Ti	RGB Tensor	$3 \times 224 \times 224$	196	72.2
DeiT-Ti*	RGB Tensor	$3 \times 224 \times 224$	196	74.35
BF-Ti (Ours)	fHWC	150528	9407	77.06
	fCHW	150528	9407	74.65
	TIFF	150668	9415	77.33
	PNG	150864	9428	74.94
	JPEG	48564	12140	65.92

Table 1. ImageNet Top-1 accuracy of ByteFormer Tiny (BF-Ti) using various file encodings, compared to DeiT-Ti. $\mathbb{E}[S]$ denotes the input shape, and $\mathbb{E}[L_t]$ denotes the token length passed to the transformer backbone. (*) denotes our implementation of DeiT-Ti. We set BF-Ti's Conv1D kernel size to $k = 32$ for all experiments except JPEG ($k = 8$).

4.1.2 ByteFormer

We describe our ByteFormer model for inference on file bytes. An overview of our model is given in Figure 2. The main challenge in using file bytes as inputs is the long sequence lengths. In Table 1, we observe that input sizes $\mathbb{E}[S]$ for various file encodings can exceed $150,000$ tokens. As described below, we use strided Conv1D and shifted window attention [22] to handle long sequence lengths.

The first step of our model is to use a learned token embedding with a vocabulary size of 256 (corresponding to 2^8 unique byte values) to produce embeddings. This choice allows our model to handle a variety of input modalities.

The next step of our model is to perform a Conv1D to reduce the sequence length. Our intuition for choosing Conv1D is that neighboring file bytes often contain related information. Reducing our sequence length with Conv1D greatly improves memory usage. In Table 1, $\mathbb{E}[L_t]$ denotes the input size to our Transformer, which is significantly smaller than $\mathbb{E}[S]$. Typically, we set our kernel size $k = 32$. Our stride is always $k/2$.

Next, we add positional embeddings to the token embeddings, then pass our embeddings to a Transformer. We choose Transformer size parameters to match the 12-layer DeiT-Ti [38] architecture with embedding dimension 192. We call this particular version of our architecture ByteFormer Tiny (BF-Ti). To compensate for our long sequence length (9417 for TIFF, compared to 196 for DeiT-Ti), we use shifted window attention [22] to limit the attention window size w, alleviating the quadratic complexity of attention layers on sequence length. We also add down-sampling layers to halve the sequence length, as in [22]. We add them after transformer blocks 0, 1, 3, 5, 7, and 9. After passing our tokens through the transformer, we average the embed-

dings across the sequence dimension.

4.2. Inference on Obfuscated Inputs

ByteFormer is designed to perform inference on file encodings without converting them into a standard input representation (e.g. an RGB tensor in the case of images). Therefore, we explore whether ByteFormer can be used for inference on privacy-preserving representations that obfuscate information about the underlying data (Figure 1c).

Consider a permutation $\phi : \{0, 1, 2, \ldots, 255\} \rightarrow \{0, 1, 2, \ldots, 255\}$. Let τ denote a token embedding, and let f_θ denote the subsequent transformer. It's easy to see that, for a given ϕ, there exists a $\tau_{\phi^{-1}}$ such that $\tau_{\phi^{-1}}(\phi(x)) = \tau(x)$. $\tau_{\phi^{-1}}$ is simply a copy of τ with embedding vectors reassigned to different indices. Thus, $f(\tau_{\phi^{-1}}(\phi(x))) = f(\tau(x))$. The implication of this statement is that our network f_θ can operate on re-encoded inputs $\phi(x)$ *without requiring any retraining* as long as the network's token embedding τ is reordered to $\tau_{\phi^{-1}}$.

To take advantage of this property, we choose a permutation ϕ at random before training. All training and inference inputs are remapped using ϕ. We optionally apply uniform noise before applying ϕ. Without uniform noise, ϕ can be applied to a standard ByteFormer without retraining (as explained above). However, we find uniform noise helpful in obfuscating regions of constant color in our experiments.

More generally, we can use more sophisticated methods for altering input representations. As our method can handle highly nonlinear JPEG encodings, we expect it to perform well on a variety of alternative encodings that an outside observer might not be able to easily guess. How secure are such methods against an adversary? This analysis depends on the threat model used. For example, if an adversary has access to a large number of encoded samples $\phi(x)$, analysis of byte statistics might suggest that strings of common bits correspond to patches of blue sky in images. The adversary's task is made more difficult given certain file encodings (e.g. the highly nonlinear JPEG encoding). We do not make strong claims regarding the level of security provided by different choices of ϕ. Secure systems should be designed and analyzed by security researchers. Instead, we simply suggest that decoupling the input representation from the model can lead to new possibilities for building more secure systems.

4.3. Privacy-Preserving Camera

We describe another application of ByteFormer to privacy-preserving inference (Figure 1d). In this scenario, a custom camera captures a non-standard, privacy-preserving representation to allow for inference without building a full RGB image. This custom representation could take a variety of forms. In our experimentation, we consider a hypothetical camera that masks out a large fraction of its pixel channels. The camera stores the remaining unmasked pixel channels in an array without retaining the coordinates of pixel channels on the image sensor. In this scenario, an adversary could not obtain a faithful reconstruction of the input image. Even if the adversary could guess pixel channel locations, the low resolution of captured data prevents the adversary from recovering a high-fidelity image.

5. Experiments

We evaluate ByteFormer on 1000-way classification on ImageNet [6]. We also evaluate 12-way audio keyword classification (including "background" and "unknown" classes) of 1-second audio clips sampled at 16 khz using Speech Commands v2 [42]. For all experiments, ByteFormer's backbone uses hyperparameters that match DeiT-Ti [38]. We refer to this architecture as BF-Ti.

We train using CVNets [27]. For ImageNet, we use batch size 48 on 8 NVIDIA A100 GPU machines. At training time, we use random resized cropping, random horizontal flipping, RandAugment [5], and RandomErase [45] before storing the image in the desired file encoding (subsubsection 4.1.1). We train with AdamW [23] with weight decay 0.05, and a cosine annealing learning rate schedule from 0.001 to 0.00002, with 7500 warmup iterations.

We train our Speech Commands v2 with MixUp [44], noise augmentation, and time shifting augmentation, as in [29]. Our training and architecture hyperparameters match our ImageNet experiments. We train these models on 4 NVidia A100 GPU machines.

For ImageNet experiments, we report Top-1 accuracy of models trained with exponential moving average of weights with momentum 0.0001, which typically increased accuracy by roughly 0.25%. For Speech Commands V2, we found EMA to sometimes increase and sometimes decrease accuracy, so we omit it.

5.1. ImageNet File Encodings

Table 1 summarizes results for a variety of file encodings on the ImageNet dataset. For BF-Ti, we use $w = 128$ and $k = 32$ for all models except JPEG, for which we find $k = 8$ to perform better. Our method surpasses DeiT-Ti accuracies for TIFF, PNG, fCHW, and fHWC encodings.

We find training on JPEG to be more difficult. This is likely due to the highly nonlinear and variable-length JPEG encoding. We investigate the influence of our model's kernel size k on JPEG accuracy in Table 2. We find that reducing k from its default value of 32 increases accuracy. Since JPEG images have a smaller token length than TIFF or PNG, they are likely less compressible. To further explore this, we investigate two settings for JPEG quality factor in Table 2. We find that lower quality factors result in lower token lengths, thus reducing k improves accuracy. We also try reducing w, but accuracy does not improve.

q	w	k	$\mathbb{E}[S]$	**Top-1**
100	128	32	48564	60.86
100	128	16	48564	64.86
100	128	8	48564	65.92
60	128	32	8436	31.8
60	128	16	8436	50.11
60	128	8	8436	56.26
60	128	4	8436	62.52
60	32	32	8436	37.23
60	32	16	8436	50.24
60	32	8	8436	56.74
60	32	4	8436	59.52

Table 2. ImageNet Top-1 accuracy for ByteFormer Tiny (BF-Ti) for different JPEG quality factors q, window sizes w, and convolutional kernel sizes k. $\mathbb{E}[S]$ denotes the expected shape of the inputs during validation.

Model	**Input**	w	k	$\mathbb{E}[S]$	**Top-1**
BC-ResNet-8	log Mel	-	-	40×98	98.70
BF-Ti (Ours)	W-FP32	128	32	64058	95.80
		128	16	64058	95.51
BF-Ti (Ours)	W-INT32	128	32	64044	94.90
		128	16	64044	95.27
BF-Ti (Ours)	W-INT16	128	32	32044	94.81
		128	16	32044	95.51
		128	8	32044	95.13
BF-Ti (Ours)	W-UINT8	128	32	16044	92.28
		128	16	16044	94.39
		128	8	16044	94.81
		128	4	16044	93.99
BF-Ti (Ours)	MP3	128	8	3465	88.39
		128	4	3465	88.00
		32	8	3465	88.69
		32	4	3465	89.19

Table 3. Results for audio classification with BF-Ti on the Speech Commands v2 dataset. "W-" denotes WAV files with the given bit width. $\mathbb{E}[S]$ denotes the shape of network inputs.

We present our method's computational efficiency compared to related works in Appendix A.

5.2. Speech Commands v2 File Encodings

Results for audio classification on Speech Commands v2 [42] are given in Table 3. BF-Ti achieves accuracies of up to 95.51% on WAV files, comparable to the state-of-the-art method BC-ResNet-8 [18]. Note that BC-ResNet-8 is specifically designed for audio processing. By contrast, we performed no parameter tuning relative to our ImageNet training recipe (besides ablating choices of w and k). Our best-performing model has $w = 128$ and $k = 32$. Our model performs best on floating-point values. In this case, since each 32-bit floating-point value in the audio signal will be encoded as 4 file bytes, each audio sample will be represented by 4 neighboring tokens before our Conv1D.

We investigate the influence of k on model accuracy. In general, the optimal k decreases when the expected number of input tokens decreases. This matches our observations in ImageNet JPEG experiments. For MP3 files, we observed that $k = 32$ resulted in unstable models due to the drastic reduction in token length. For MP3, we additionally experiment with $w = 32$, but it does not improve results.

Noise level	**Model**	
	DeiT-Ti	**BF-Ti**
None	51.61	**77.39**
$\mathbb{U}[-5, 5]$	50.77	**77.27**
$\mathbb{U}[-10, 10]$	49.50	**77.17**
$\mathbb{U}[-20, 20]$	43.84	**76.31**

Table 4. ImageNet Top-1 results for obfuscation with ϕ. We show results with no noise, and with uniform noise in $[-a, a]$ added. We use the fHWC encoding.

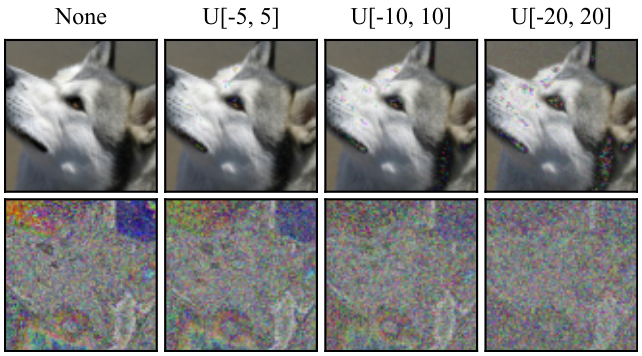

Figure 3. A sample image from the ImageNet validation set, with uniform noise applied (top row), and with byte remapping ϕ additionally applied (bottom row).

5.3. Image Obfuscation

Results for our image obfuscation method (subsection 4.2) on ImageNet are summarized in Table 4. After obtaining our fHWC encoding, we apply a randomly chosen obfuscation function ϕ.

Examples of obfuscated images are shown in Figure 3. We observe that byte remapping retains shape information. A region of the image that is dominated by a single pixel value will continue to be dominated by a new (remapped) pixel value. To alleviate this, we add noise from a uniform distribution $\mathbb{U}[-a, a]$ sampled from $-a$ to a (inclusive) to each pixel channel independently, then compute the result modulo 256. Afterwards, we apply ϕ. This prevents regions of constant pixel value from being remapped to a single value. As shown in Figure 3, the upper right corner of

Kept	75%	50%	25%	10%	5%	3%
k	32	16	8	4	4	4
Top-1	74.77	75.36	74.04	71.35	68.11	64.15

Table 5. ImageNet Top-1 accuracy for our privacy-preserving camera experiment with BF-Ti when the given fraction of pixel channels are kept.

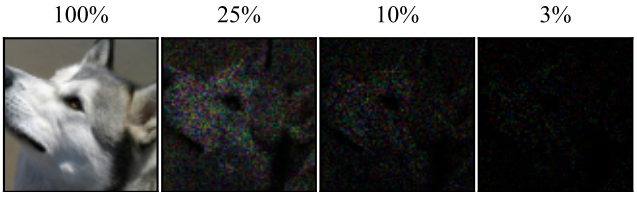

Figure 4. An ImageNet validation image captured by our hypothetical privacy-preserving camera in which the given fraction of pixel channels are kept. Note that positions of retained pixel channels is discarded by the camera. To make visualization possible, we include the positional information implicitly by placing unmasked pixels in the correct position.

the image becomes less recognizable as noise from progressively larger ranges is used. In Table 4, we observe that our method is resilient to this transformation, but DeiT is not.

5.4. Privacy Preserving Camera

Table 5 summarizes our results for our privacy-preserving camera (subsection 4.3). We emulate the camera setup by masking pixel channels of ImageNet images at random, then storing unmasked pixels in a buffer (in fHWC order) and passing that buffer into our network. For these experiments, we cannot provide DeiT-Ti baselines because DeiT-Ti is not capable of ingesting pixel values without any indication of their placement in the image.

In Figure 4, we show masked inputs before the unmasked pixels are rasterized. At 10% pixel retention, the content of image is hard to visually perceive *even though active pixels are placed side-by-side in a new buffer*. Even if an adversary correctly guessed the positions of unmasked pixel channels in the original image, the adversary could not former a high-fidelity image. As shown in Table 5, our accuracy at 10% pixel retention is 71.35%, comparable to the original DeiT-Ti model operating on non-privacy-preserving (unmasked) images.

Note that this privacy-preserving technique can be combined with the byte remapping technique (subsection 4.2) to further obfuscate network inputs.

6. Analysis

Alternate Attention Methods: We study three state-of-the-art self-attention methods for handling long sequence lengths in Transformers: (1) full self-attention [39, 7, 20] where each token attends on every other token, (2) shifted window attention [22, 1] where tokens are divided into local windows and each local window computes self-attention independently, and (3) bag (or hierarchical) attention [28, 3] where tokens are broken up into bags and each bag computes intra-bag self-attention. Inter-bag self-attention is then computed on the resultant output. These different methods are visualized in Figure 5 while results on TIFF data are summarized in Table 6. We choose TIFF for these experiments because of its long sequence length (Table 1). We find window attention to outperform bag attention. Note that full attention cannot be run due to its $\mathcal{O}(n^2)$ dependence on sequence length n. In our main experiments, we

Attention	**Top-1**
Full	OOM
Window	77.33
Bag	75.20

Table 6. ImageNet Top-1 accuracy of BF-Ti with different types of attention. We run out of memory with full attention.

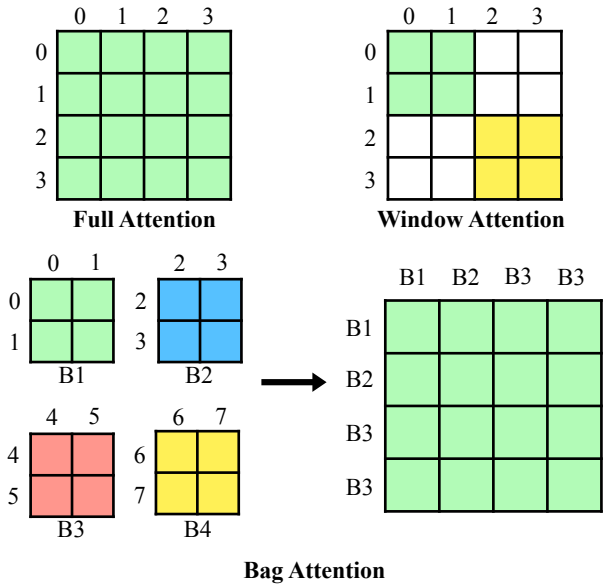

Figure 5. Illustration of the types of attention we experiment with. Bag attention is computed in two stages. First, individual bags compute attention. Then, attention is computed across bags.

Augmentation	**Top-1**
Random Shuffle	3.06
Stride	5.64
Window Shuffle	18.14
Cyclic	60.97
Reverse	61.23
Baseline	60.81

Table 7. Ablation showing the Top-1 ImageNet accuracy of BF-Ti on JPEG images ($k = 32$, quality factor 100). See text for details.

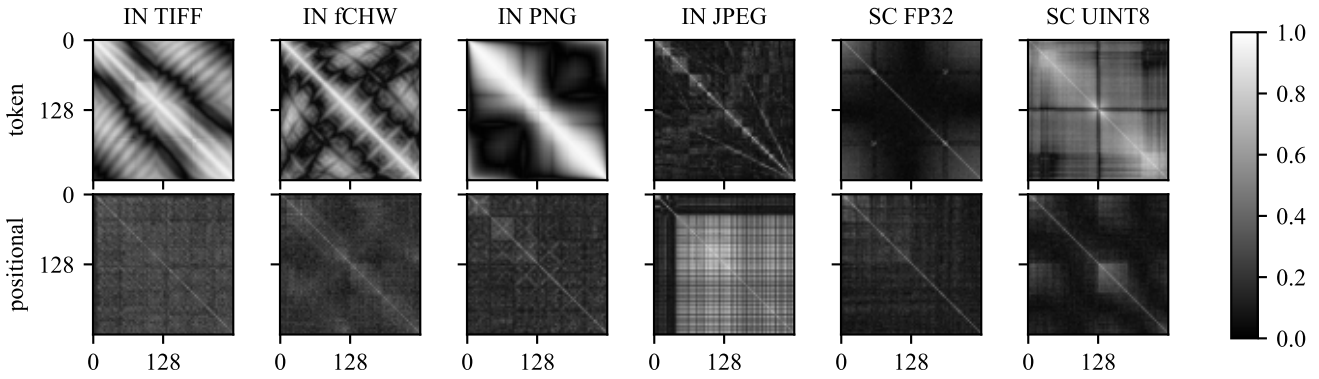

Figure 6. $|x \cdot y|/(||x|| \cdot ||y||)$ for pairs x, y of token embeddings (top row) and positional embeddings (bottom row) learned by BF-Ti. We show results for various file encodings on ImageNet (IN) and Speech Commands (SC).

used shifted window attention.

Effect of Byte Ordering: To better understand ByteFormer's behavior, we ask, *does ByteFormer simply learn byte frequencies, or is the byte ordering relevant?* In Table 7, we apply a series of augmentations during training and validation. We focus on the case of JPEG compression at quality factor 100 with our standard kernel size $k = 32$. Each augmentation modifies the byte order of the inputs in some way. In `random shuffle`, we randomly reorder the bytes during training and validation. The order is redrawn every iteration. This severely degrades accuracy. Next, we perform a strided sampling with stride size 1024 (e.g. $[0, 1024, 2048, \ldots, 1, 1025, 2049, \ldots]$). This slightly improves accuracy over the previous method by improving byte order consistency. Next, we experiment with `window shuffle`, in which the bytes from each window of size 1024 are consistently permuted. This increases accuracy to 18.14%. Next we experiment with a `cyclic` shift in which the second half of the image bytes are moved to the beginning. Accuracy matches the baseline (unaltered JPEG bytes) closely. Similarly, `reverse`, in which the byte order is reversed, preserves locality well and matches the baseline. We find that our model is sensitive to locality, and does not only learn byte frequencies.

Learned Token Embeddings: We study the token embeddings learned by ByteFormer. These embeddings are used to project file bytes into vector representations. In Figure 6 (top row), we observe the absolute value of the cosine distance $|x \cdot y|/(||x|| \cdot ||y||)$ between each pair of token embeddings x, y on a variety of file encodings. We choose this metric to highlight the difference between (anti-)correlated embeddings (bright patches) and uncorrelated embeddings (dark patches). The pattern varies substantially across input encodings and tasks. In TIFF, PNG, and fCHW, we observe a bright band off of the diagonal, corresponding to high correlation between bytes and their neighbors. This matches our expectations, since replacing a byte with its neighbor only slightly alters the image. This does not hold for JPEG due to the Huffman encoding step. We also observe that the correlation between token embeddings in the float32 encoding of Speech Commands is generally weak. We believe this occurs because the float32 audio amplitude value is split across four bytes in the file encoding, weakening the association between byte values and amplitudes.

Learned position embeddings We visualize the absolute value of the cosine distance between the first 256 positional embeddings learned by ByteFormer in Figure 6 (bottom row). For JPEG, we see a strong band of highly uncorrelated values at early positions, corresponding to the file header. Later positions demonstrate interesting patterns that may arise due to the Huffman encodings crossing byte boundaries. In TIFF, a small band of highly uncorrelated values is visible early on, corresponding to the header (which is shorter than in the JPEG case).

7. Limitations

The accuracy of ByteFormer depends on the file encoding chosen. As shown in section 5, choosing JPEG over TIFF results in a reduction of accuracy on ImageNet. Adding invariance to file encodings is future work.

As discussed in subsection 4.2, our choice of ϕ for our obfuscation method does not provide cryptography-level security against an attacker with access to a large set of model inputs. We view this method as a building block for security experts to design thoroughly analyzed, secure systems.

Finally, our method has only been evaluated on classification for images and audio. Experimenting with other domains (video, text) and tasks that require fine-grained localization (detection, segmentation) is exciting future work.

8. Conclusion

We present ByteFormer, a model that consumes only bytes and does not explicitly model the input modality. We show that it achieves strong performance on image and audio classification without hyperparameter tuning or architecture modifications. We show how ByteFormer can be used in conjunction with image obfuscation techniques with little or no loss in accuracy. We also demonstrate how ByteFormer can be incorporated into a privacy-preserving camera to enable inference without forming a full image at capture time.

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[2] T. Boutell. Png (portable network graphics) specification. `https://www.rfc-editor.org/rfc/rfc2083`. Accessed: 2023-03-05.

[3] Richard J Chen, Chengkuan Chen, Yicong Li, Tiffany Y Chen, Andrew D Trister, Rahul G Krishnan, and Faisal Mahmood. Scaling vision transformers to gigapixel images via hierarchical self-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16144–16155, 2022.

[4] Alex Clark. Pillow (pil fork) documentation, 2015.

[5] Ekin Dogus Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3008–3017, 2019.

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

[7] 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*, abs/2010.11929, 2020.

[8] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer, 2013.

[9] Graham M. Gibson, Steven D. Johnson, and Miles J. Padgett. Single-pixel imaging 12 years on: a review. *Opt. Express*, 28(19):28190–28208, Sep 2020.

[10] Yuan Gong, Yu-An Chung, and James R. Glass. Ast: Audio spectrogram transformer. *ArXiv*, abs/2104.01778, 2021.

[11] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[13] Catherine Higham, Roderick Murray-Smith, Miles Padgett, and Matthew Edgar. Deep learning for real-time single-pixel video. *Scientific Reports*, 8, 02 2018.

[14] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Andrew Brock, Evan Shelhamer, Olivier J. H'enaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs. *ArXiv*, abs/2107.14795, 2021.

[15] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, 2021.

[16] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: Fss-based secure training with gpus. Cryptology ePrint Archive, Paper 2023/206, 2023. `https://eprint.iacr.org/2023/206`.

[17] Peter Kabal. Wave file specifications. `https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html`. Accessed: 2023-03-05.

[18] Byeonggeun Kim, Simyung Chang, Jinkyu Lee, and Dooyong Sung. Broadcasted residual learning for efficient keyword spotting. In *Interspeech*, 2021.

[19] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 336–353, 2019.

[20] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*, pages 280–296. Springer, 2022.

[21] Hao Liu, Wilson Yan, and Pieter Abbeel. Language quantized autoencoders: Towards unsupervised text-image alignment, 2023.

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

[24] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. *ArXiv*, abs/2206.08916, 2022.

[25] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. Accessed: 2023-03-06.

[26] Dave Marshall. The discrete cosine transform. https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html. Accessed: 2023-03-06.

[27] Sachin Mehta, Farzad Abdolhosseini, and Mohammad Rastegari. Cvnets: High performance library for computer vision. *Proceedings of the 30th ACM International Conference on Multimedia*, 2022.

[28] Sachin Mehta, Ximing Lu, Donald L. Weaver, Joann G. Elmore, Hannaneh Hajishirzi, and Linda G. Shapiro. Hatnet: An end-to-end holistic attention network for diagnosis of breast biopsy images. *ArXiv*, abs/2007.13007, 2020.

[29] Dianwen Ng, Yunqi Chen, Biao Tian, Qiang Fu, and Chng Eng Siong. Convmixer: Feature interactive convolution with curriculum learning for small footprint and noisy far-field keyword spotting. *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3603–3607, 2022.

[30] M. Nilsson. The audio/mpeg media type. `https://www.rfc-editor.org/rfc/rfc3003.html`. Accessed: 2023-03-05.

[31] Jeongsoon Park and Justin Johnson. Rgb no more: Minimally-decoded jpeg vision transformers. *ArXiv*, abs/2211.16421, 2022.

[32] G. Parsons and J. Rafferty. Tag image file format (tiff). `https://www.rfc-editor.org/rfc/rfc3302`. Accessed: 2023-03-05.

[33] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[34] Vijay Raghunathan. Ece264: Huffman coding. https://engineering.purdue.edu/ece264/17au/hw/HW13?alt=huffman. Accessed: 2023-03-07.

[35] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *27th Annual Conference on Computer and Communications Security (ACM CCS 2020)*. ACM, August 2020.

[36] James Robert, Marc Webbie, et al. Pydub, 2018.

[37] Akash Shah, Nishanth Chandran, Mesfin Dema, Divya Gupta, Arun Gururajan, and Huan Yu. Secure featurization and applications to secure phishing detection. In *Proceedings of the 2021 on Cloud Computing Security Workshop*, CCSW '21, page 83–95, New York, NY, USA, 2021. Association for Computing Machinery.

[38] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2020.

[39] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[40] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[41] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019:26 – 49, 2019.

[42] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *ArXiv*, abs/1804.03209, 2018.

[43] Wikipedia. Jpeg. `https://en.wikipedia.org/wiki/JPEG`. Accessed: 2023-03-05.

[44] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ArXiv*, abs/1710.09412, 2017.

[45] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI Conference on Artificial Intelligence*, 2017.

A. Performance

Our goal is to directly model file bytes, with privacy-preserving applications. Since all previous methods involve some level of domain-specific modeling at inference time (including file decoding and different stems for different input domains), direct comparison disadvantages our model. Nevertheless, it's important to characterize the runtime of our model, and compare to previous approaches. This helps to contextualize our model's performance.

We compare BF-Ti to related works in Table 8. We show performance on ImageNet [6], including efficiency and accuracy metrics. We only report train time and throughput for models we trained ourselves. This is to avoid hardware differences creating inconsistent results. For these experiments, we tuned BF-Ti's batch sizes to maximize GPU utilization. Note that this improved training time by a relatively small amount (less than 10%) compared to the experiments in section 5.

Our model's size and accuracy (8.82 million parameters, 77.27%) falls between DeiT-Ti (5.72 million parameters, 78.62%) and DeiT-S (22.05 million parameters, 73.20%). Our model's forward pass time is slower due to the large number of tokens being modeled. Domain-specific modeling (which our model avoids) can drastically reduce compute time.

Compared to other multi-modal models [15, 14], our model achieves a comparable accuracy with far fewer flops and a far smaller model. Note that our model performs no

Model	M	$\mathbb{E}[L_t]$	**Top-1**	Sec	P (M)	F (B)	Im/s
MobileNetv3 Large	✘	N/A	75.1	-	5.43	0.22	9615
ResNet-50	✘	N/A	78.12	-	25.55	4.017	3488
DeiT-S p16	✘	196	78.62	336	22.05	4.61	3594
DeiT-Ti p=16	✘	196	73.20	334	5.72	1.26	6885
DeiT-Ti p=14	✘	256	74.62	331	5.69	1.70	4970
DeiT-Ti p=8	✘	784	77.44	824	5.72	7.06	1243
RGB No More DeiT-Ti [31]	✘	196	75.1	-	5.72	1.26	6885
Perceiver (learned pos) [15]	✔	N/A	67.6	-	55.9	62.3	-
Perceiver IO (learned pos) [14]	✔	N/A	72.7	-	62.3	407	-
Perceiver (conv) [15]	✔	N/A	77.4	-	42.1	367	-
Perceiver IO (conv) [14]	✔	N/A	82.1	-	48.6	369	-
BF-Ti k=32	✔	9415	77.27	1314	8.82	23.74	373
BF-Ti k=32 -C	✔	9415	74.54	1122	7.64	12.63	370
BF-Ti k=32 -C -NPE	✔	9415	68.42	1121	5.83	12.63	372
BF-Ti k=4 f0.05	✔	3762	67.53	368	6.70	5.70	1687
BF-Ti k=4 f=0.1	✔	7524	71.26	580	7.42	11.07	875
BF-Ti k=8 f=0.25	✔	9407	73.65	769	7.93	15.40	634

Table 8. ImageNet Top-1 accuracy. **M**: whether the model accepts various modalities (✘: No. ✔: Yes, but with modality-specific modeling. ✔: Yes). $\mathbb{E}[L_t]$: length of token inputs to transformer (after Conv1D for BF-Ti. Note, Perceiver feeds inputs through cross-attention, so this concept doesn't directly apply). **Sec**: Train epoch time (only reported for models we train to avoid hardware differences affecting results). **P (M)**: Number of parameters (millions). **F (B)**: Number of flops (billions). **Im/s**: Throughput (images/sec) on an A100 80GB Nvidia GPU. "-" means "not reported". For DeiT, p is patch size. Choosing $p \leq 4$ is unfeasible (epochs take hours or days). For BF-Ti, k is conv kernel size, and f indicates fraction of retained pixels for privacy-preserving camera experiments (subsection 5.4). "-C" indicates replacement of Conv1D with a windowed mean. "-NPE" indicates an ablation that removes the positional embedding.

domain-specific modeling at inference time. By contrast, Perceiver includes domain-specific modeling (file decoding, tensor reshaping, and optionally convolutions) at inference time.