ANALYZING DEFORESTATION IN THE AMAZON BASIN WITH MACHINE LEARNING

SANTANIL JANA, MANISH KRISHAN LAL, YOUSSEF MOUSAAID, KIMATHRA REDDY, REZA SADOUGHINAN, AND C YU

Abstract. Using machine learning, we analyze Amazon basin deforestation data. Inspired by the 2016 Kaggle competition "Planet: Understanding the Amazon from Space," we enhance standard CNNs like ResNet50 with tensor network techniques, reducing parameters and complexity. Our contribution involves replacing traditional CNNs with "Tensorized" CNNs, achieving improved performance. This project presents a benchmarking analysis comparing Tensorized CNNs to conventional ones for Amazon basin deforestation tracking.

1. Introduction

Every minute, the world loses an expanse of forest equivalent to 48 football fields, with the Amazon Basin standing as a primary contributor to this alarming statistic. The consequences of deforestation in this critical region extend far beyond the loss of trees, impacting biodiversity, habitats, climate, and unleashing devastating effects globally. In response to this environmental crisis, Planet, a visionary creator of the world's most extensive constellation of Earth-imaging satellites, embarked on a groundbreaking initiative to revolutionize the way we perceive and combat deforestation.

The central challenge in monitoring and understanding deforestation dynamics lay in the limitations on the efficiency of existing methods. While substantial research efforts have been invested in tracking changes in forests, the reliance on coarse-resolution imagery from satellites like Landsat and MODIS has hindered precision, particularly in areas dominated by small-scale deforestation. Moreover, distinguishing between human-induced and natural causes of forest loss remained a persistent challenge.

In this context, Planet, in collaboration with its Brazilian partner SCCON, initiated a Kaggle competition, inviting data scientists and machine learning enthusiasts to contribute their expertise (see [2]). The objective was to label satellite image chips, providing insights into atmospheric conditions and various classes of land cover/land use in the Amazon Basin. This competition marked a pivotal step towards harnessing the power of high-resolution satellite imagery, collected daily by Planet's constellation, to generate a nuanced understanding of where, how, and why deforestation occurs worldwide.

The set-up of Kaggle's labeled database is as follows. The goal is for the network to apply one or more suitable labels to each satellite image. There are 17 labels

in total separated into three categories: four atmospheric conditions (clear, partly cloudy, cloudy, haze), seven common land cover and land use types (rainforest, agriculture, rivers, towns/cities, roads, cultivation, bare ground), and six rare land cover and land use types (slash and burn, selective logging, blooming, conventional mining, artisanal mining, blow down). Thus, an image may be labeled simply "cloudy" as the cloudiness interferes with our ability to determine the land cover and land use type, or an image may be labeled both "partly cloudy" and "agriculture" at the same time.

The competition posed a unique set of challenges, including the inherent noise in chip labels due to the ambiguity of features and the complexities introduced by cloud cover. Clouds, a formidable obstacle in passive satellite imaging, presented a major hurdle in monitoring the Amazon Basin. To address this, participants were tasked with labeling atmospheric conditions, common land cover/land use phenomena, and rare occurrences. The resulting algorithms were poised to become invaluable tools for governments and local stakeholders, enabling quicker and more effective responses to deforestation and human encroachment on forests.

While not part of the official competition, we tackle the same problem and take an approach partly inspired by but different from the solutions on Kaggle. In particular, we introduce and apply the technique of tensorization to pre-existing Convolutional Neural Networks (CNNs) in order to decrease the number of parametres and the complexity.

2. Methods

Our main method involved modifying and applying a tensorization technique, Tensor Train (TT) Decomposition, to the preexisting ResNet-50 neural network. In this section we lay out the mathematical foundation and our set-up for the experiments.

We assume the reader has some familiarity with the most basic models of Neural Networks and Convolutional Neural Networks (CNNs), involving an input layer, at least one hidden layer, and an output layer. In a high level description, the standard type of neural network uses matrix-vector multiplication and vector addition between layers, and training this network means minimizing the loss (cost) function by manipulating the various weights and biases involved, using techniques such as gradient descent. A Convolutional Neural Network (CNNs)'s hidden layers, however, uses an inner-product-like operation between matrices. Minimization processes are similar and further subtleties exist, including the requirement of a Fully Connected (FC) layer which connects every neuron in one layer to every neuron in another layer. For image classification purposes, a CNN is commonly used because it better captures the nuances of images while remaining robust under modifications such as translation of parts of images.

2.1. **ResNet-50.**

In this section, we discuss ResNet-50 and review its architecture. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer) that was introduced in the 2015 paper "Deep Residual

Learning for Image Recognition" by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian (see [3]). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks and combines depth and scale for performance.

The 50-layer ResNet architecture includes the following elements:

- A 7×7 kernel convolution alongside 64 other kernels with a 2-sized stride.
- A max pooling layer with a 2-sized stride.
- 9 more layers—3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times.
- 12 more layers with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.
- 18 more layers with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times.
- 9 more layers with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times.
  This makes 50 layers up to this point.
- Average pooling, followed by a fully connected layer with 1000 nodes, using the softmax activation function.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

FIGURE 1. Architecture of ResNet-50.

For the majority of our project we work based off of the pre-existing network ResNet-50. The depth of the network enables the capturing of complex features and avoids the vanishing gradient problem. The challenge with ResNet-50, however, lies in its size. ResNet-50 has over 25 million parameters, resulting in long training times and large computations. Since massive computation resources are needed, not only is time and efficiency of research a concern, it might also contribute more significantly to an increased carbon footprint.

2.2. **Tensor Train Decompositions.** Tensor decomposition models compress tensors that stores parameters of a neural network model. Tensor decomposition extends the concept of matrix decomposition to higher-order tensors. There are various tensor decomposition methods such as Canonical-Polyadic (CP) decomposition, Tucker decomposition and Tensor-Train decomposition. See for example [6] for more details about these techniques. In this project, we apply Tensor-Train decomposition, which is a powerful technique in which original vectors and matrices are now represented and analyzed as high-dimensional tensors, and breaking down multi-dimensional arrays into simpler components (ranks) offers computation efficiency. Specifically, in order to achieve this, we will use Tensor Trains (TT).

We briefly review the Tensor-Train decomposition. Our reference is [1] and [4]. A d-dimensional array (tensor) $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \times \mathbb{R}^{n_d}$ is said to be represented in the TT-format if for each dimension $k = 1, ..., d$ and for each possible value of the k-th dimension index $i_k = 1, \ldots, n_k$, there exists a matrix $G_k[i_k]$ such that all the elements of $\boldsymbol{\mathcal{T}}$ can be computed as the following matrix product:

$$\boldsymbol{\mathcal{T}}(i_1, i_2, \ldots, i_d) = \mathbf{G}_1[i_1]\mathbf{G}_2[i_2] \ldots \mathbf{G}_d[i_d] \ . \tag{2.1}$$

All the matrices $G_k[i_k]$ related to the same dimension k are restricted to be of the same size $r_{k-1} \times r_k$. The values r_0 and r_d are equal to 1 in order to keep the matrix product (2.1) of size 1×1. We refer to the representation of a tensor in the TT-format as the *TT-representation* or the *TT-decomposition*. The sequence $\{r_k\}_{k=0}^{d}$ is referred to as the *TT-ranks* of the TT-representation of $\boldsymbol{\mathcal{T}}$ (or the ranks for short) and its maximum as the maximal TT-rank of the TT-representation of $\boldsymbol{\mathcal{T}}$ denoted by r, i.e. $r = \max_{\{k=0,\ldots,d\}} r_k$. The collections of the matrices $\{\mathbf{G}_k[i_k]\}_{i_k=1}^{n_k}$ corresponding to the same dimension (technically, 3-dimensional arrays G_k) are called the *cores* and denoted by $\boldsymbol{\mathcal{G}}_k$, i.e. the tensor $\boldsymbol{\mathcal{T}}$ has d number of cores $\boldsymbol{\mathcal{G}}_1$, $\boldsymbol{\mathcal{G}}_2$,...,$\boldsymbol{\mathcal{G}}_d$ where each $\boldsymbol{\mathcal{G}}_k \in$ is a 3-dimensional array $\mathbb{R}^{n_k} \times \mathbb{R}^{r_{k-1}} \times \times \mathbb{R}^{r_k}$.

We use the symbols $G_k[i_k](\alpha_{k-1}, \alpha_k)$ to denote the element of the matrix $G_k[i_k]$ in the position (α_{k-1}, α_k), where $\alpha_{k-1} = 1, ..., r_{k-1}$ and $\alpha_k = 1, ..., r_k$. Equation (2.1) can be equivalently rewritten as the sum of the products of the elements of the cores:

$$\boldsymbol{\mathcal{T}}(i_1, i_2, \ldots, i_d) = \sum_{\alpha_0=1}^{r_1} \sum_{\alpha_1=1}^{r_2} \cdots \sum_{\alpha_d=1}^{r_d} \mathbf{G}_1[i_1](\alpha_0, \alpha_1)\mathbf{G}_2[i_2](\alpha_1, \alpha_2) \ldots \mathbf{G}_d[i_d](\alpha_{d-1}, \alpha_d) \ . \tag{2.2}$$

Computational advantage. *The TT-format requires* $\sum_{s=1}^{n} n_s r_{s-1} r_s$ *parameters to represent* $\boldsymbol{\mathcal{T}}$ *which has* $\prod_{s=1}^{n} n_s$ *elements. We want to approximate a given tensor* $\boldsymbol{\mathcal{T}}$ *by a tensor* $\tilde{\boldsymbol{\mathcal{T}}}$ *that is in TT-format. Each core* $\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{nd_k} \times \mathbb{R}^{r_{k-1}} \times \mathbb{R}^{r_k}$ *of* $\tilde{\boldsymbol{\mathcal{T}}}$ *needs* $O(nr^2)$ *parameters, where* $n = max\{n_1, .., n_k\}$ *so the overall train has* $O(ndR^2)$ *parameters (where* $d = max\{d_1, .., d_k\}$*). This is a huge improvement from the exponential number of parameters* $O(n^d)$ *required to store the original tensor* $\boldsymbol{\mathcal{T}}$*.*

This seems too good to be true, but fortunately, for most tensors, we can always choose r to be constant with respect to n.

For an arbitrary tensor $\boldsymbol{\mathcal{T}}$ a TT-representation exists but is not unique. The ranks among different TT-representations can vary and it's natural to seek a representation with the lowest ranks.

Definition 2.2. *The TT-rank of a tensor $\boldsymbol{\mathcal{T}}$ is the smallest tuple $(r_1, \cdots, r_{n-1})$ such that $\boldsymbol{\mathcal{T}}$ admits a TT decomposition with cores of sizes $r_1, \cdots, r_{n-1}$.*

We will denote by $\text{rank}(\boldsymbol{\mathcal{T}})$ *the maximum of the elements of the TT-rank, i.e.* $\text{rank}(\boldsymbol{\mathcal{T}}) = \max(r_1, r_2, \ldots, r_{n-1})$.

For a tensor $\boldsymbol{\mathcal{T}}$, and for a fixed $\epsilon > 0$ (considered as accuracy), we want to find a tensor $\tilde{\boldsymbol{\mathcal{T}}}$ that has tensor decomposition such that

$$||\boldsymbol{\mathcal{T}} - \tilde{\boldsymbol{\mathcal{T}}}||_F < \epsilon ||\boldsymbol{\mathcal{T}}||_F \ ,$$

where $||.||_F$ is the Frobenuis norm, i.e. the sum of absolute values of entries of the tensor. In other words, $\boldsymbol{\mathcal{T}}$ is approximated by a $\tilde{\boldsymbol{\mathcal{T}}}$ that has a TT-decomposition.

The TT-decomposition of tensor $\boldsymbol{\mathcal{T}}$ with cores of sizes $r_1, \cdots, r_{n-1}$ can be computed using an algorithm rooted in the singular value decomposition (SVD) technique. Refer to page 2301, algorithm 1 TT-SVD, in [5] for comprehensive details.

Remark. The TT-ranks r_k $(k = 1, 2, ..., d)$ control the trade-off between the number of parameters versus the accuracy of the representation: the smaller the TT-ranks, the more memory efficient the TT-format is.

2.3. **Tensorizing Fully-Connected (FC) Layers.** The main reference for this section is [4]. We call the *TT-layer* of a neural network to be a fully connected layer (also a dense layer) with the weight matrix stored in the TT-format. We will refer to a neural network with one or more TT-layers as *TensorNet*.

Fully-connected layers apply a linear transformation to an N-dimensional input vector x:

$$y = Wx + b, \tag{2.3}$$

where the weight matrix $W \in \mathbb{R}^{M \times N}$ and the bias vector $b \in \mathbb{R}^M$ define the transformation. A TT-layer consists in storing the weights W of the fully-connected layer in the TT-format, allowing to use hundreds of thousands of hidden units while having moderate number of parameters.

We first explain how we can compress vectors and matrices with tensor-train decomposition. The direct application of the TT-decomposition to a matrix (2-dimensional tensor) is indeed the same as compressing it using the low-rank matrix format and similarly, the direct TT-decomposition of a vector is equivalent to explicitly storing its elements. To be able to efficiently work with large vectors and matrices, their TT-format is defined in a special manner.

Consider a vector $b \in \mathbb{R}^N$ and assume $N = \prod_{k=1}^{d} n_k$. We first rewrite the vector b in the format of a d-dimensional tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1} \times ... \times \mathbb{R}^{n_d}$. We can establish a bijection

μ between the coordinate $\ell \in \{1, ..., N\}$ of b and a d-dimensional vector-index $\mu(\ell) = (\mu_1(\ell), ..., \mu_d(\ell))$ of the corresponding tensor $\boldsymbol{\mathcal{B}}$, where $\mu_k(\ell) \in \{1, ..., n_k\}$.

The tensor $\boldsymbol{\mathcal{B}}$ is then defined by the corresponding vector elements: $\boldsymbol{\mathcal{B}}(\mu(\ell)) = b_\ell$. Building a TT-representation of $\boldsymbol{\mathcal{B}}$ allows us to establish a compact format for the vector b and we refer to it as a *TT-vector*.

Now assume W is a matrix $\mathbb{R}^{M\times N}$ where $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$. Note that the matrix W is not restricted to be square and M and N can be different but we can always express them as the product of d numbers $m_1, .., m_d$ for M and $n_1, ..., n_d$ for N, respectively. We can build a TT-representation of M similar to how we did it with vector b. First, let bijections $v(t) = (v_1(t), ..., v_d(t))$ and $\mu(\ell) = (\mu_1(\ell), ..., \mu_d(\ell))$ map row and column indices t and ℓ of the matrix W to the d-dimensional vector-indices whose k-th dimensions are of length m_k and n_k respectively, $k = 1, ..., d$. From the matrix W we can form a d-dimensional tensor $\boldsymbol{\mathcal{W}}$ whose k-th dimension is of length $m_k n_k$ and is indexed by the tuple $(v_k(t), \mu_k(\ell))$. The tensor $\boldsymbol{\mathcal{W}}$ can then be converted into the TT-format as follows:

$$\begin{aligned} W(t, \ell) &= \boldsymbol{\mathcal{W}}\big((v_1(t), \mu_1(\ell)), ..., (v_d(t), \mu_d(\ell))\big) \\ &= G_1[v_1(t), \mu_1(\ell)]G_2[v_2(t), \mu_2(\ell)]...G_d[v_d(t), \mu_d(\ell)], \end{aligned}$$

where the matrices $G_k[v_k(t), \mu_k(\ell)]$, $k = 1, .., d$ serve as the cores with tuple $\big(v_k(t), \mu_k(\ell)\big)$ being an index.

A TT-layer transforms a d-dimensional tensor $\boldsymbol{\mathcal{X}}$ (formed from the corresponding vector x) to the d-dimensional tensor Y (which correspond to the output vector y). We assume that the weight matrix W is represented in the TT-format with the cores $G_k[i_k, j_k]$ as shown above. The linear transformation (2.3) of a fully-connected layer can be expressed in the tensor form:

$$Y(i_1, ..., i_d) = \sum_{j_1,...j_d} \Big(G_1[i_1, j_1]...G_d[i_d, j_d]\Big)\boldsymbol{\mathcal{X}}(j_1, ..., j_d) + \boldsymbol{\mathcal{B}}(i_1, ..., i_d). \tag{2.4}$$

Direct application of the TT-matrix-by-vector operation for the Eq. (2.4) yields the computational complexity of the forward pass $O(dr^2 m \max\{M, N\})$ (where $m = \max\{m_1, .., m_d\}$).

2.4. **Tensorizing Convolutional Layers.** The idea is to motivate a particular way of applying the Tensor-Train format to the convolutional kernel. We can rethink convolutional layer as matrix-matrix multiplication so that we can apply similar tensor-train decomposition as of Fully-connected layer. Convolutional layer transforms the 3-dimensional input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{W\times H\times C}$ into the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{(W-\ell+1)\times(H-\ell+1)\times S}$ by convolving $\boldsymbol{\mathcal{X}}$ with the *kernel tensor* $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{\ell\times\ell\times C\times S}$:

$$\mathcal{Y}(x, y, s) = \sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\sum_{c=1}^{C} \mathcal{K}(i, j, c, s)\mathcal{X}(x + i - 1, y + j - 1, c). \tag{2.5}$$

To improve the computational performance, one can reduce the convolution (2.5) to a matrix-by-matrix multiplication as follows: We first denote $H' = H - \ell + 1$ and

$W' = W - \ell + 1$. we then reshape the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{W' \times H' \times S}$ into a matrix Y of size $W'H' \times S$ in the following way:

$$\mathcal{Y}(x, y, s) = Y(x + W'(y - 1), S).$$

Then $\mathbf{Y}$ can be decomposed into the product of two matrices $\mathbf{X} \in \mathbb{R}^{W'H' \times \ell^2 C}$ and $\mathbf{K} \in \mathbb{R}^{\ell^2 C \times S}$:

$$\mathbf{Y} = \mathbf{X}\mathbf{K},$$

where

$$\mathcal{X}(x + i - 1, y + j - 1, c) = X(x + W'(y - 1), i + \ell(j - 1) + \ell^2(c - 1)),$$

and

$$\mathcal{K}(i, j, c, s) = K(i + \ell(j - 1) + \ell^2(c - 1), s),$$

for $y = 1, ..., H'$, $x = 1, ..., W'$, $i, j = 1, ..., \ell$.

3. Results: Compression of Existing CNN Models

3.1. **ResNet50.** The main reference for this section is [1]. In this experiment, we're exploring the optimization of the tensor rank in the context of a Convolutional neural network model. The setup involves a baseline model consisting of a ResNet50 architecture followed by a fully connected layer (FC1) with dimensions $(2048, 512)$, resulting in a substantial number of parameters (over 1 million). The goal here is to investigate how different tensor ranks in the TT decomposition of this FC1 layer impact both model accuracy and training time.

We apply the method of transfer learning by replacing the output layer of ResNet-50 with 17 neurons (corresponding to our 17 labels), and we obtain 32768×17 new weight matrix parameters and 17 bias vector parameters, totalling 557073 new parameters. Upon first experimentation, retraining only the final layer of this new model for 20 epochs with Google Colab GPU takes around 40 minutes. We wish to improve this.

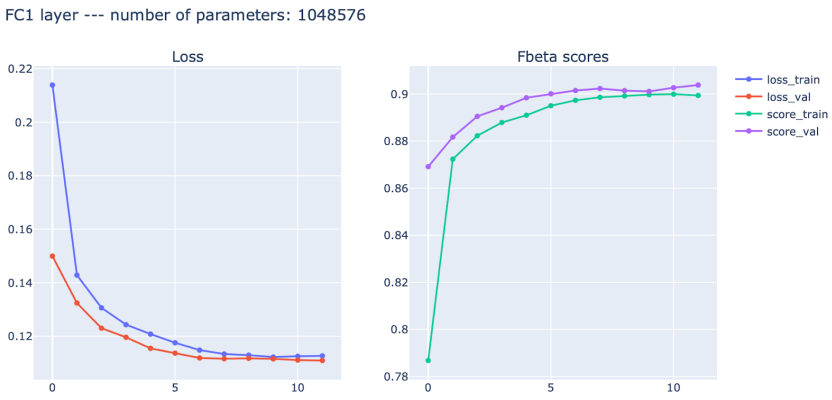

Figure 2. Fully Connected Layer 1 performance.

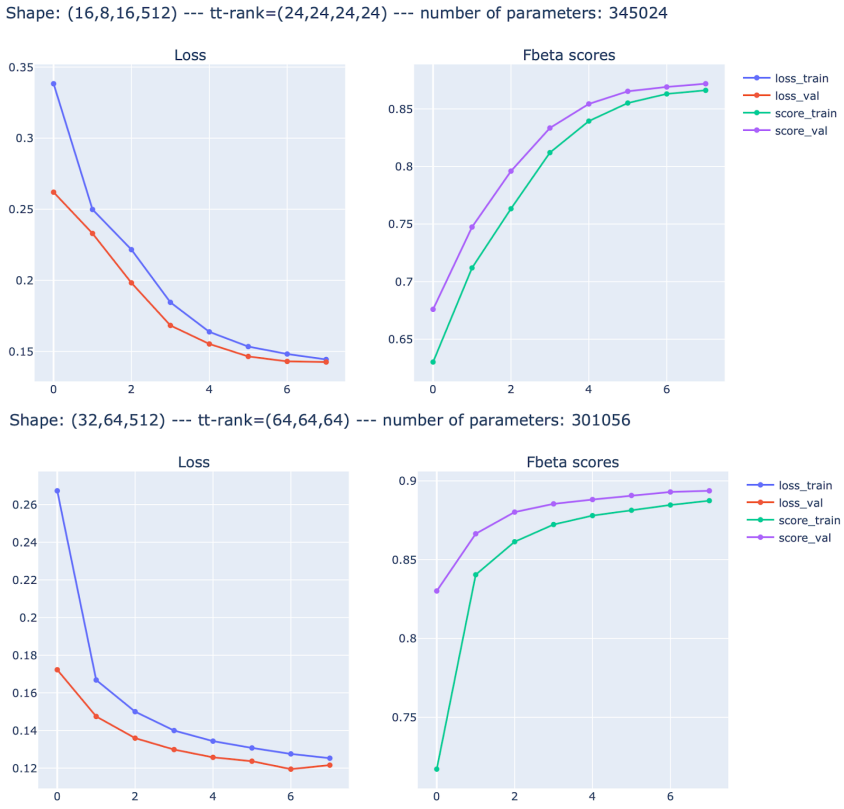

FIGURE 3. Two different tensorizations.

Using TT decomposition on the weight matrix on the output layer of the ResNet-50 architecture, we are able to reduce the number of parametres to only 17425. Using this new model, retraining in the same set up takes less time and gives comparable accuracy. We can see that tensorizing the entire model using TT decomposition gives significant advantages in terms of speeding up training time and utilization of resources. Complete tensorization using TT decomposition will further reduce training time and model size, enabling implementation on previously inaccessible devices.

The experiment begins by reshaping the weight matrix of the FC1 layer, which has dimensions 2048×512, and then applying the tensor train (TT) decomposition to this weight matrix. TT decomposition represents the weight matrix as a sequence of smaller tensors, thus reducing the number of parameters and the computational complexity of the layer. Various TT-rank configurations are tested to determine their effect on model performance.

The experimental results are insightful. The baseline ResNet50 model, while accurate (90% accuracy), is computationally intensive, taking 929 seconds per epoch. When the FC1 layer is tensorized with an $(8\times8\times8\times8)$ configuration, the compression ratio increases significantly $(1 : 12)$, leading to a trade-off with accuracy (71%) but

only a slight increase in training time (859 seconds per epoch). On the other hand, a TT-rank of $(16 \times 16 \times 16 \times 16)$ maintains a good accuracy rate (83%) while staying computationally efficient (851 seconds per epoch). This suggests that the TT decomposition can effectively compress the FC1 layer while preserving model accuracy.

Model	**Comp. ratio**	**Accuracy**	**Epoch (sec)**
Baseline ResNet18 + FC	1:1	89.00%	847
$(8 \times 8 \times 8)$, tt-rank=(8,8,8,8)	1:12	71.00%	859
$(8 \times 8 \times 8)$, tt-rank=(16,16,16,16)	1:1	83.12%	851
Baseline ResNet50 + FC	1:1	90.30%	929
$(16 \times 8 \times 16)$, tt-rank=(24,24,24,24)	1:3	87.20%	935
$(16 \times 8 \times 16)$, tt-rank=(32,32,32,32)	1:1	89.09%	939
(32×64), tt-rank=(64,64,64)	1:3.5	89.37%	929

TABLE 1. Comparison of compression ratio and accuracy for various tensorization and tt-ranks applied on ResNet50 .

In the case of ResNet50 with a $(16 \times 8 \times 16)$ tensorization, the TT-rank configuration of $(24 \times 24 \times 24 \times 24)$ provides a good balance between compression $(1:3)$ and accuracy (87%) while maintaining a similar training time (935 seconds per epoch) to the baseline. These results highlight the potential of tensorization techniques to significantly reduce model size and computational requirements while achieving competitive accuracy. Further experimentation and optimization of tensor ranks may offer even more efficiency and scalability in deep learning models, making them more accessible for resource-constrained environments. See table 1 and plots shown in Figures 2 and 3 above for more details.

3.1.1. *A simple CNN model.* We implemented tensor train composition on a compact CNN model that includes four convolutional and two dense layers (see [7]). The diagram below (Figure 4) illustrates its architecture.

The model exhibits a training accuracy of 0.9054 and a validation accuracy of 0.9060. Specifically, we applied TT-decomposition to compress the last dense layer of this CNN architecture, experimenting with various TT-ranks $(r_0, r_1, ...)$ and various max tt-rank $= \max_{i=1,...,d,\ i_k=1,...n_k} \{\text{rank}(G_i[i_k])$. Surprisingly, in all cases, the accuracy remained

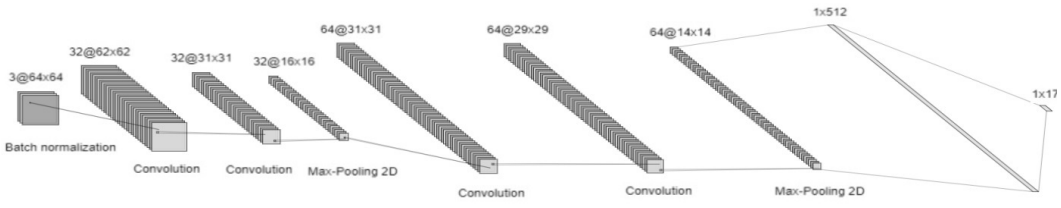

FIGURE 4. The architecture of the CNN model of section 3.1.1.

consistent with the existing CNN model. This implies that the compressed models required fewer parameters while achieving identical training and validation (test) accuracy. Refer to the table below 2 for the detailed experimental results.

TT-decomposition of the last dense layer with 8721 parameters				
TT-ranks	max tt-rank	#params of comp layer	train accuracy	test accuracy
$[8, 8, 8]$	8	1518	0.9054	0.9060
$[8, 8, 8]$	1	6	0.9054	0.9060
$[2, 2, 128]$	1	2197	0.9054	0.9060
$[128, 2, 2]$	1	181	0.9054	0.9060
$[32, 8, 2]$	2	181	0.9054	0.9060

TABLE 2. Comparison of the number of parameters and train and test accuracies for various tensorization and tt-ranks applied on this the CNN model of section 3.1.1 .

The table 2 illustrates that for TT-ranks $[8, 8, 8]$ and max TT-rank=1, the number of parameters in the compressed layer decreased significantly from 8721 to just 6, while maintaining consistent train and validation (test) accuracies.

Exploring tensor-train decomposition on other layers of this network is an intriguing possibility, but we defer this to future work.

4. Conclusion

Big neural networks traditionally take a large amount of computational resources to train. There is also a lot of redundancy in big neural networks, adding to the inefficiency of the computational stress. We have seen that tensorization allows the reduction of the size of a Fully Connected layer of a CNN with only a small drop in accuracy. The ultimate goal is to increase accuracy while decreasing memory and complexity. With the right tensorization (i.e. choice of dimensions of arrays) we observed that we may get comparable accuracy with significantly less variables. Future research should continue to explore the qualities of tensorization and behaviors of different dimensions in each tensorization in order to truly optimize this process.

5. Acknowledgement

The authors would like to thank the Pacific Institute for the Mathematical Sciences and the Math to Power Industry program for giving us the opportunity to work on this project. We are also grateful to the team at Multiverse, in particular Sukhi Singh and Mehdi Bozzo-Rey, for their guidance and insight and support on this project.

References

1. Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry P. Vetrov, *Ultimate tensorization: compressing convolutional and FC layers alike*, CoRR **abs/1611.03214** (2016).

2. Benjamin Goldenberg, Burak Uzkent, Christian Clough, Dennis Funke, Deven Desai, grischa, JesusMartinezManso, Kat Scott, Meg Risdal, Mike Ryan, Pete, Rachel Holm, Ramesh Nair, Sean Herron, Tony Stafford, and Wendy Kan, *Planet: Understanding the amazon from space*, 2017.
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, CoRR **abs/1512.03385** (2015).
4. Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov, *Tensorizing neural networks*, Advances in Neural Information Processing Systems (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.
5. I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing **33** (2011), no. 5, 2295–2317.
6. Yannis Panagakis, Jean Kossaifi, Grigorios G. Chrysos, James Oldfield, Mihalis A. Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou, *Tensor methods in computer vision and deep learning*, Proceedings of the IEEE **109** (2021), no. 5, 863–890.
7. Christian Tan_PH, *Satellite image classification with cnns: P, a, e.*, 2023.

University of British Columbia
E-mail address: santanil@math.ubc.ca

University of British Columbia
E-mail address: manish.krishanlal@ubc.ca

Perimeter Institute
E-mail address: ymousaaid@perimeterinstitute.ca

University of British Columbia
E-mail address: kimathra@gmail.com

University of British Columbia
E-mail address: reza.sadoughinan@gmail.com

University of Toronto
E-mail address: cyu@math.toronto.edu