Exploration of Vision Transformers and CNNs: In-Depth Structural and Cost Analysis, Model Performance, and Applications in Biomedical Images

ENM5310 Project - Jeffrey Li, Vaibhav Wanere

Abstract:

The objective of this project was two-fold. Firstly, we wanted to achieve greater understanding of Vision Transformers (ViTs) and Convolutional Neural Networks (CNNs) by implementing them from scratch and studying their architectures and core components. From this, we implemented the ViT from *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* through following a tutorial [1]. We assessed the performance of the model with varying input patch sizes on the CIFAR-10 dataset. Our trials demonstrated that a patch size of 4 was ideal for this problem. When compared with CNNs, ViT achieved better performance results with a small number of epochs but suffered from high training computational runtime. Secondly, we wanted to be able to apply the material we learned from our exploration and devise a model to address real-world problems. We selected the RSNA Screening Mammography Breast Cancer Detection dataset from Kaggle for this problem. From our results, we see the model achieved approximately 60% accuracy on the image dataset, which may be attributed to insufficient amount of training images.

Section I: Introduction and Problem Statement

Convolutional neural networks (CNN) are currently the dominant architecture for computer vision. Recent success of self-attention-based architectures for natural language processing (NLP) inspired the implementation of Vision Transformers (ViT) for image classification tasks [1]. According to the original paper by Dosovitskiy et al., ViT achieves comparable or even superior performance to convolutional architectures, namely ResNet, in image classification in *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [1]. Although transformers lack some of the inductive biases inherent in CNNs, Dosovitskiy et al. overcame these shortcomings through training on larger datasets ranging from 14 to 300 million images. From the group's paper, ViTs with classification (CLS) tokens demonstrated strong preservation of spatial information and showed high promise for future uses in object detection [2].

Through this work, we take an explorative approach to understanding the vision transformer architecture by addressing the following problems:

1. What are the process and structural differences between traditional CNNs and Vision Transformers?
2. How can we understand model performances of Vision Transformers?
3. Can we use Vision Transformers to solve biomedical image classification tasks? [3]

We implemented a vision transformer from scratch using JAX following the original paper by Dosovitskiy et al. as well as through seeking inspiration and aid from tutorials and online resources. Additionally, due to computational resources limitations and time constraints, we decided to train and evaluate our model on the CIFAR-10 dataset [4]. We acknowledged it would still be valuable to understand how vision transformers perform given a much smaller dataset compared to the ones used in

the original paper. For our application, we wanted to see how vision transformers performed on biomedical images, in which image image data may be imbalanced, so we worked with the RSNA Screening Mammography Breast Cancer Detection dataset from Kaggle [3]. Analyzing structural differences and understanding how ViT processes images through internal representations will pave the way for further advancements in other computer vision problems and applications. The Vision Transformer is a promising new deep learning architecture with potential to surpass CNNs in computer vision tasks.

The rest of the paper is organized accordingly. We first provide a background on transformer models and architecture in **Section II**, before providing a visual walkthrough of how Visual Transformers process image data in **Section III**. In **Section IV**, we detail our work on an implementation of a Vision Transformer from scratch, and in **Section V**, we report and interpret our results after training and evaluating our model on the CIFAR-10 dataset. In **Section VI**, we discuss our results after training and evaluating our model on RSNA Screening Mammography Breast Cancer Detection dataset from Kaggle. This is lastly followed by a discussion of improvements to the project and future work in **Section VII**.

Section II: Transformers Background

A. Transformers Components

Transformers were first proposed by Vaswani et al. in *Attention Is All You Need* for machine translation tasks, and have since become a widely popular tool in natural language processing (NLP) problems [1]. This new type of architecture was introduced to address many of the shortcomings of Recurrent Neural Network-based models in processing and analyzing sequential text data [5].

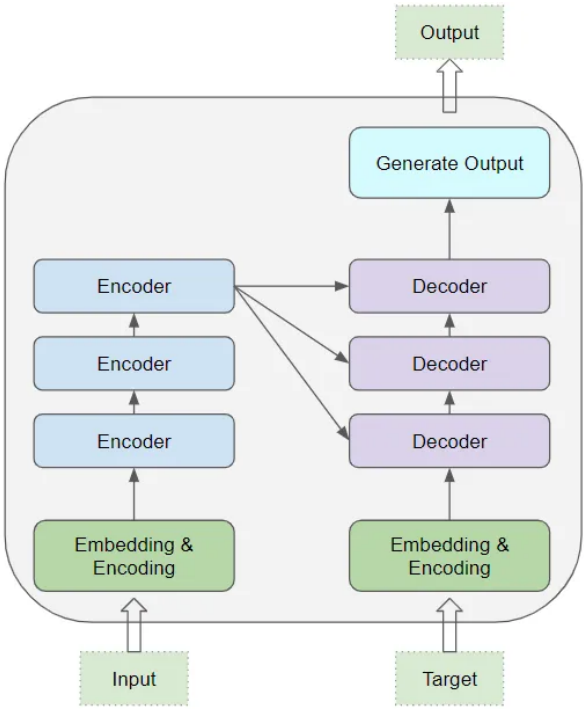


Figure One: Transformer Architecture [6].

A simple transformer architecture contains a stack containing multiple Encoder layers and a stack containing multiple Decoder layers [6]. The **Encoder** stack enables the model to capture the contextual relationships between words and tokens in the input text. On the other hand, the **Decoder** stack takes the output of the final Encoder layer and uses it to generate a sequence of tokens [6]. Each Encoder and Decoder stack contains a layer for embedding (which converts input text into a vector) and positional encoding (which provides information about the position of each token or word in an input sequence) [5, 6].

Transformer's adoption of the **Attention Mechanism** allowed for performance beyond those of previously dominant recurrent neural network-based models. Both the Encoder stack and the Decoder stack possess the Self-Attention Layer whereas the Decoder stack contains the Encoder-Decoder Attention layer [5, 6]. The self-attention layer enables the model to focus on other words in the input that are closely related to that word [6]. The Encoder-Decoder Attention layer allows the decoder to selectively attend to different parts of the encoder output when generating the output sequence.

B. Mathematical Definition

The attention mechanism can be represented mathematically as a **Scaled Dot-Product**, in which Q represents the query vector (used to describe the current input token), K represents the key vector (used to identify the token of interest in the query), and V represents the value vector (used to compute the output for each token) [5].

$$Attention(Q,\ K,\ V)\ =\ softmax(\frac{QK^T}{\sqrt{d_k}})\mathrm{V}$$

Equation One: Scaled Dot-Product Extension [5].

In short, the attention mechanism helps the model focus on certain components of an input sequence when processing the information. Firstly, we perform the dot product between Q and K to obtain the attention scores [5]. The resulting values measure the similarity between the current input token and all other tokens in the sequence [5]. Scaling the attention scores by the dimensions of k is used for stabilizing the gradient during training.

The scaled scores are passed through a softmax function to obtain a probability distribution over the tokens in the sequence. This is followed by a matrix multiplication operation with V to obtain a weighted sum as the attention output. This output captures the most relevant parts of the input sequence and is then passed through other layers of the neural network [5].

Multi-Head Attention is an extension of Scaled Dot-Product Attention, in which linear transformations are run on the queries, keys, and values to yield multiple sets of inputs to the attention function [5]. The attention function is computed in parallel given each of these sets of inputs, and the results are then concatenated side-by-side to form one matrix [5]. The final result is obtained via a linear transformation of the concatenated matrix to obtain a matrix with the desired dimensionality.

The intuition behind multi-head attention is that it allows us to attend to different parts of the sequence at different times. Through this, the model can better capture positional information because each head will attend to different input segments. Moreover, each head will also capture different contextual information by uniquely correlating words [7].

Section III: Vision Transformer Model Explored

A. Vision Transformer Architecture

This section aims to provide more information regarding the ViT model from the original paper. Dosovitskiy et al. adopt the Transformer architecture proposed by Vaswani et al for image classification tasks [1]. For their model input, images are divided into a grid of square patches. Each patch is then flattened into a single vector by concatenating the channels of all pixels in a patch and then linearly projecting it to the desired input dimension [1,7]. Positional embeddings are added to each patch to the model to learn about the structure of the images [1]. This is because traditional transformers are incapable of extracting and modeling spatial information from data [7].

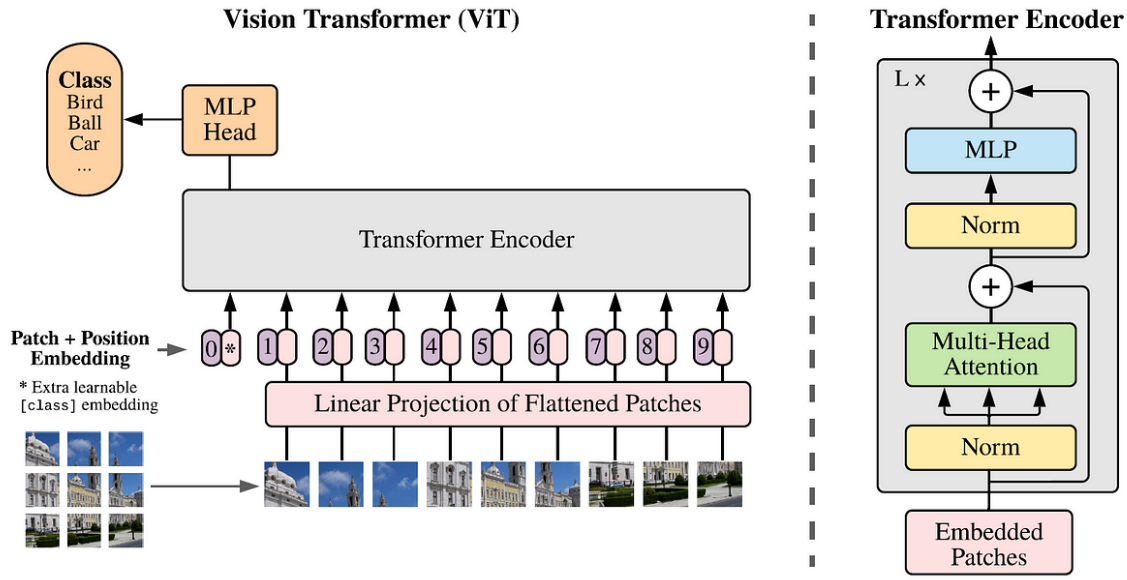


Figure Two: ViT Architecture and Encoder Layer [1].

For performing classification tasks, an extra learnable "classification token" (CLS token) is added to the patch embedding vectors [1, 7]. CLS tokens are widely used in NLP classification problems to represent the target class that the model is attempting to predict [6]. The embedded vectors are fed into a standard Encoder layer. As previously mentioned in **Section II**, the Encoder contains the Self-Attention Layer (Multi-Head Attention) which is used to capture contextual information about the input sequence [5]. Layer normalization standardizes the activations across the features dimension of a layer. Lastly, the features are passed through a multi-layer perceptron (MLP) consisting of fully-connected (FC) linear

layers which allow the model to learn global contextual information and positional information [7]. Predictions of class probabilities are outputted [7].

Inside Look at Vision Transformer

We wanted to explore the data flow of the Vision Transformer as well as what the intermediate steps looked like. The objective is to obtain a better understanding of how Vision Transformers work "inside the hood." We used a tutorial Colaboratory notebook curated by Hiroto Honda for this task [8]. The notebook details step-by-step of the Vision Transformer process. We report some visualizations through the figures below.

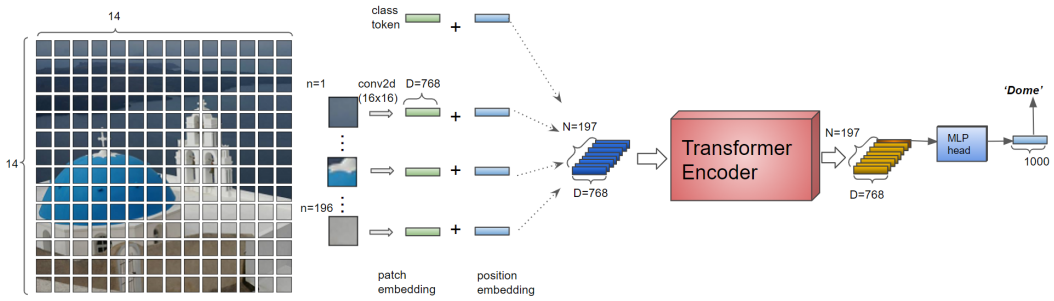


Figure Three: Vision Transformer data flow.

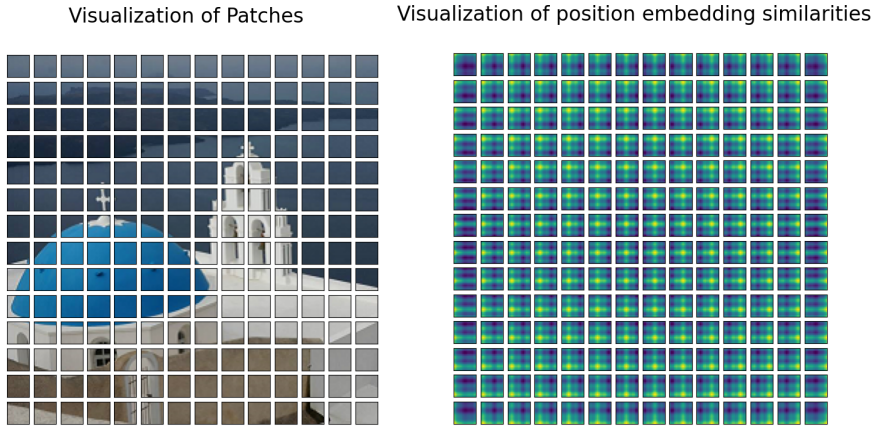


Figure Four (left): Image is divided into smaller, equal-sized patches.
Figure Five (right): Each path has its own positional embedding, which allows the model to learn about structural and positional information in an image.

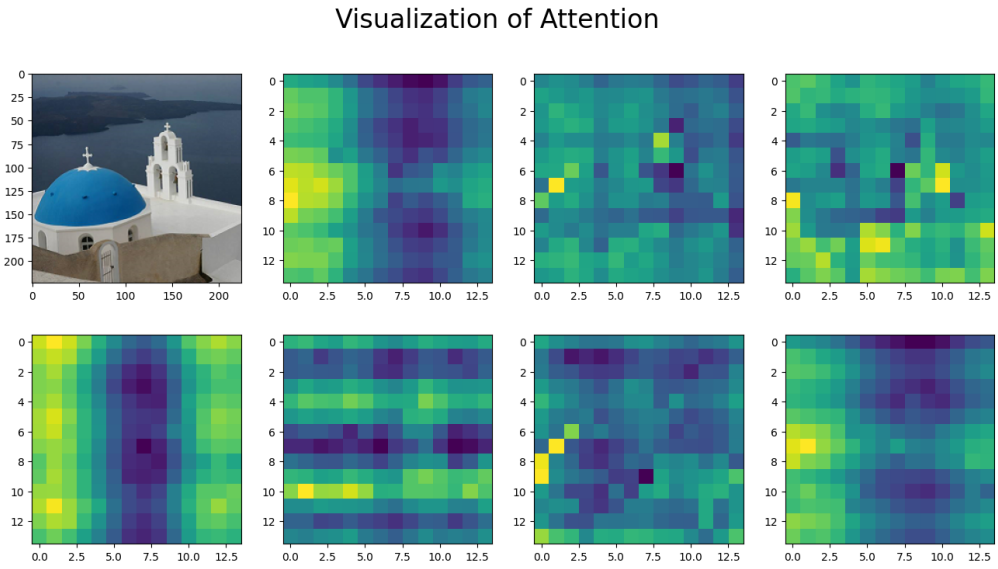


Figure Six: One attention score matrix computed from seven heads of the multi-headed attention layer.

Section IV: Vision Transformer Implementation and Design Decisions

We followed a tutorial curated by Phillipe Lippe for implementing the Vision Transformer using JAX [9]. The implementation modifies the original ViT model through different design decisions. We detail the implementation in this section before performing some smaller experiments on the CIFAR-10 dataset in **Section V**.

Software/Hardware: Standard Python libraries were used, including Numpy and MatPlotLib. Additionally, JAX libraries were used, namely the jax.numpy module, for enhanced computing performance. The Flax library was used for implementing neural network layers. The Optax library was used for optimizing and updating the parameters of our models. We relied on Google Colaboratory as the main Python environment for creating and executing our models. We utilized the NVIDIA Cudafor GPU.

Data Augmentation: To accommodate for overfitting, we transform images using *RandomHorizontalFlip()*, which flips each image horizontally by a chance of 50%, and *RandomResizedCrop()*, which changes the aspect ratio before readjusting the image to the original size. We did not anticipate any changes to the image class following these transformations.

Encoder: We construct our Encoder using a Multi-Headed Attention mechanism and a linear layer. Unlike the implementation from the original paper, layer performalization was performed prior to each of these layers in order to obtain smoother gradient flow [9]. Often during training, the covariate shift in features changes and *LayerNorm* helps mitigate the effects of this phenomena by normalizing across all features and activations in a layer. We used *MultiHeadDotProductAttention*, which allows for multiple attention heads to be computed in parallel. For the linear layer, we used the GELU (Gaussian Error Linear Units) activation function to mitigate problems with vanishing gradients. GELU differs from RELU (Rectified Linear Unit) as it is smoother near zero and differentiable at all ranges, even having gradients at negative ranges. Interestingly, this is the same activation function used in the BERT model. GELU and

Dropout were both sandwiched between two fully connected layers (*Dense*). All of these were implemented using modules and functions from Flax.

Vision Transformer Model: The rest of the Vision Transformer was implemented in a similar manner from the original paper. For the final *MLPHead*, *LayerNorm* was performed on the inputs prior to passing through a fully-connected layer and outputting the probability distribution for ten classes.

Optimizer Design: For our optimizer, we used Optax's *adamw* which is ADAM with weight decay, similar to the original paper implementation. Weight decay is useful for regularizing learning towards small weights. Gradient clipping (Optax's *clip_by_global_norm*) was used to mitigate the exploding gradient problem by clipping tensors by the ratio of the sum of their norms. Cross entropy loss (*softmax_cross_entropy_with_integer_labels*) was used as the loss function.

Hyperparameters: Hyperparameters were adjusted accordingly as the size of the dataset used is significantly smaller than those used in the original paper. Notable hyperparameters used include an embedding dimension of 256 which defines the size of the input and attention feature vectors, hidden dimension of 512 for the hidden layer of the fully-connected layers, the learning rate is 3e-4, the number of heads for the Multi-Headed Attention is 8, and the number of layers in the Transformers is 6.

Section V: Vision Transformer on CIFAR-10 Results

For this section, we wanted to assess the performance of the Vision Transformer model using the CIFAR-10 dataset. We experimented with different patch sizes of 2, 4, and 8 for our image input as it may impact how the model learns features from an image. We report our findings in the table and figures below. The total number of epochs was reduced to 50 as the training time for transformers tends to be lengthy.

Table 1: Vision Transformer Trials Over 50 Training Epochs

Patch Size	Training Time (m:s)	Train Accuracy	Test Accuracy	Train Loss	Test Loss
2	20:25	0.742	0.707	0.671	0.772
4	34:16	0.797	0.749	0.572	0.479
8	19:16	0.719	0.668	0.735	0.814

Figure Seven (left): Accuracy plot for experiment with patch size of four, trained over 50 epochs.
Figure Eight (right): Loss plot for experiment with patch size of four, trained over 50 epochs.

From the results, we see that a patch size of four led to better accuracy and loss at the expense of longer training time. Training accuracy reached 0.742 and testing accuracy reached about 0.707 after 50 epochs. Training loss steadily decreased over the number of epochs to around 0.572 whereas testing loss hovered around 0.5. Additional plots can be found in **Appendix A** of the **Supplementary Information**.

As a baseline model comparison, we used the CNN architecture from Assignment 4 trained over 50 epochs. The architecture for the CNN baseline is briefly described here:

- Convolution layer: 6 filters, filter size 5x5x3, stride 1
  ReLU layer
  Max pooling layer: Pooling size 2x2, stride 2
- Convolution layer: 12 filters, filter size 5x5x6, stride 1
  ReLU layer
  Max pooling layer: Pooling size 2x2, stride 2
- Convolution layer: 24 filters, filter size 5x5x12, stride 1
  ReLU layer
  Max pooling layer: Pooling size 2x2, stride 2
- Fully connected network with 2 hidden layers.

Table 2: Baseline CNN Trial Over 50 Training Epochs				
Training Time (m:s)	**Train Accuracy**	**Test Accuracy**	**Train Loss**	**Test Loss**
00:10	0.375	0.382	0.181	0.189

From the results, we see that the CNN model's performance was worse compared to the Vision Transformer's performance. This may be attributed to underfitting as the model may require more epochs to train over to successfully generalize over new data. Moreover, it is important to note that the Vision

Transformer trains over substantially more parameters which increases performance and computational runtime.

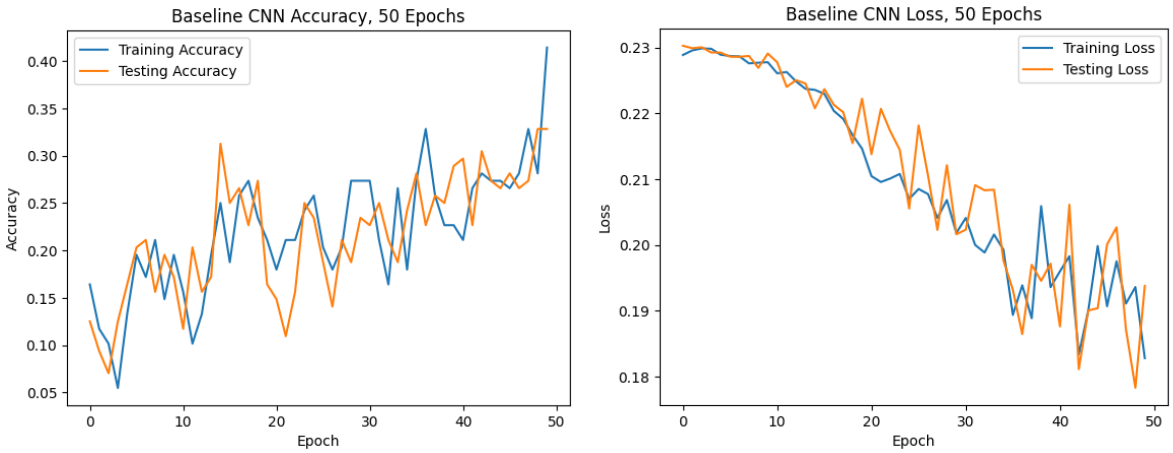


Figure Seven (left): Accuracy plot for CNN baseline, trained over 50 epochs.
Figure Eight (right): Loss plot for CNN baseline with patch size of four, trained over 50 epochs.

Section VI: Vision Transformer on Biomedical Image Data

The dataset used was from RSNA (Radiological Society of North America). The dataset was contributed by mammography screening programs in Australia and the U.S. It includes detailed labels, with radiologists' evaluations and follow-up pathology results for suspected malignancies. The ViT model has been trained on the RSNA dataset and its goal is to detect breast cancer from the images.
The image dataset is available in dicom (Digital Imaging in Medicine) file format, we first need to convert it to a numpy array which can then be used as the input to the model. For this task we used the python pydicom library.

We followed the following steps to create patch embeddings for the images which are shown in the following figure.
1, Divide the image into patches.
2. Flatten the patches to a vector.
3. Create a sequence of these vectors for a given image.
4. Map these vectors to D dimensions using trainable linear projection.
We also need to prepend a learnable embedding to the patch embeddings. State of this embedding at the output of the transformer serves as the label for image classification.
Position embeddings are added to patch embeddings to retain the positional information [Vaswani et al].

The MLP contains two layers with a GELU non-linearity.

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\cdots;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

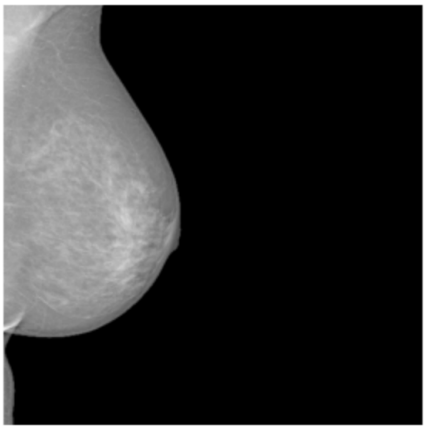


Image without patching

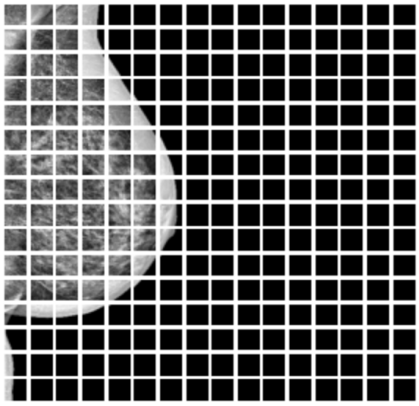


Image with patch size (32, 32)

Model Hyperparameters:

The model has following hyperparameters:

1. Image size: For our case it was (512, 512) pixels with one channel
2. Patch size (P, P) we used square patches for simplicity.
3. Model dimensions; : this is the number of features in the output of each attention head.
4. Number of attention heads.
5. Dimension of feedforward network in the transformer.
6. Blocks- number of transformer blocks in the network.
7. MLP head units in the neural network layers.
8. Number of classes- In our case, it is binary classification.

Results:

The training was performed with a batch size of 8 over 10 epochs.

The loss and accuracy for different hyperparameters is shown below:

Vision Transformer Trials Over 10 Training Epochs			
Patch Size	**Training Time**	**Accuracy**	**Loss**
32	8 min	0.48	0.69
16	20 min	0.62	0.58
8	27 min	0.54	0.65

Apart from batch size, the hyperparameters like number of attention heads, number of transformer blocks, MLP head units can be altered to test the performance of the model. Owing to computational constraints we couldn't test for all the hyperparameters.

Section VII: Discussion and Future Work

The Vision Transformer introduced by Dosovitskiy et al. is a promising new deep learning architecture with potential to surpass CNNs in computer vision tasks [1]. In this work, we wanted to explore and assess the performances of the Vision Transformer model in image classification problems. Firstly, we presented a primer on traditional transformer models for NLP before describing more about the Vision Transformer architecture and data flow. We then implemented a Vision Transformer from scratch using JAX and then trained our new model on CIFAR-10 data. The results from these experiments were compared with a baseline CNN model from Assignment 4.

From the three trials evaluated on the CIFAR-10, we varied the patch size of the images being fed into the Encoder layer. The three patch sizes were size 2, 4, and 8. Over 50 epochs, we see that the best model performance is achieved by the model trained with patch size of 4. Having a patch size too small may be too difficult for the model to learn the positional relationship between each of the patches. Moreover, having a larger patch too large may not allow the model to extract relevant information regarding certain features properly. These trials can be improved by increasing the number of epochs trained over as to mitigate the risks stemming from the bias-variance tradeoff.

When compared with the baseline CNN model, we noticed that its performance over the 50 epochs was relatively poor. This can be attributed to underfitting as the model may be of very low complexity. From the Assignment 4, we trained the same model and architecture over 20000 epochs to obtain relatively good performance. Despite this, the training time for the CNN model was substantially lower than the training time for the Vision Transformer. The reasons for these differences can be explained through the larger number of parameters that need to be trained for the later model, the number of layers in the later model, and the attention mechanism which has quadratic runtime complexity.

For future work on this project, we would intend on utilizing a cloud computing platform like Amazon Web Services to access large compute clusters to train our models on. From there, we can increase the number of epochs and perhaps even increase the dataset size we are training on. We expect to see the model performance of the Vision Transformer to be significantly better, referencing the results from the original paper by Dosovitskiy et al., because the model requires a substantially large amount of training data [1]. Interestingly, the introduction of more training data may mitigate the disadvantage that Vision Transformers have for lacking certain inductive biases regarding local connectivity. Secondly, we originally wanted to implement and assess the model performance of ResNet (2015) using the CIFAR-10 dataset, but were unable to due to time constraint. A performance assessment of ResNet and ViT on larger datasets would be worthwhile.

For the detection of cancer from biomedical image dataset, we obtain an accuracy of around 60% and thus this is not a reliable model. The reasons for this are the number of training epochs being less, the image dataset may not be sufficient for the model to learn. Related works have used CNN for the same dataset.

References

[1] An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. https://arxiv.org/abs/2010.11929.

[2] Do Vision Transformers See Like Convolutional Neural Networks? https://arxiv.org/abs/2108.08810.

[3] RSNA Screening Mammography Breast Cancer Detection. https://www.kaggle.com/competitions/rsna-breast-cancer-detection.

[4] The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[5] Transformers: a Primer. http://www.columbia.edu/~jsl2239/transformers.html.

[6] Transformers Explained Visually (Part 1): Overview of Functionality. https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452.

[7] Vision Transformer: What It Is & How It Works [2023 Guide]. https://www.v7labs.com/blog/vision-transformer-guide.

[8] An unofficial colab walkthrough of Vision Transformer. https://medium.com/@hirotoschwert/an-unofficial-colab-walkthrough-of-vision-transformer-8bcd592ba26a.

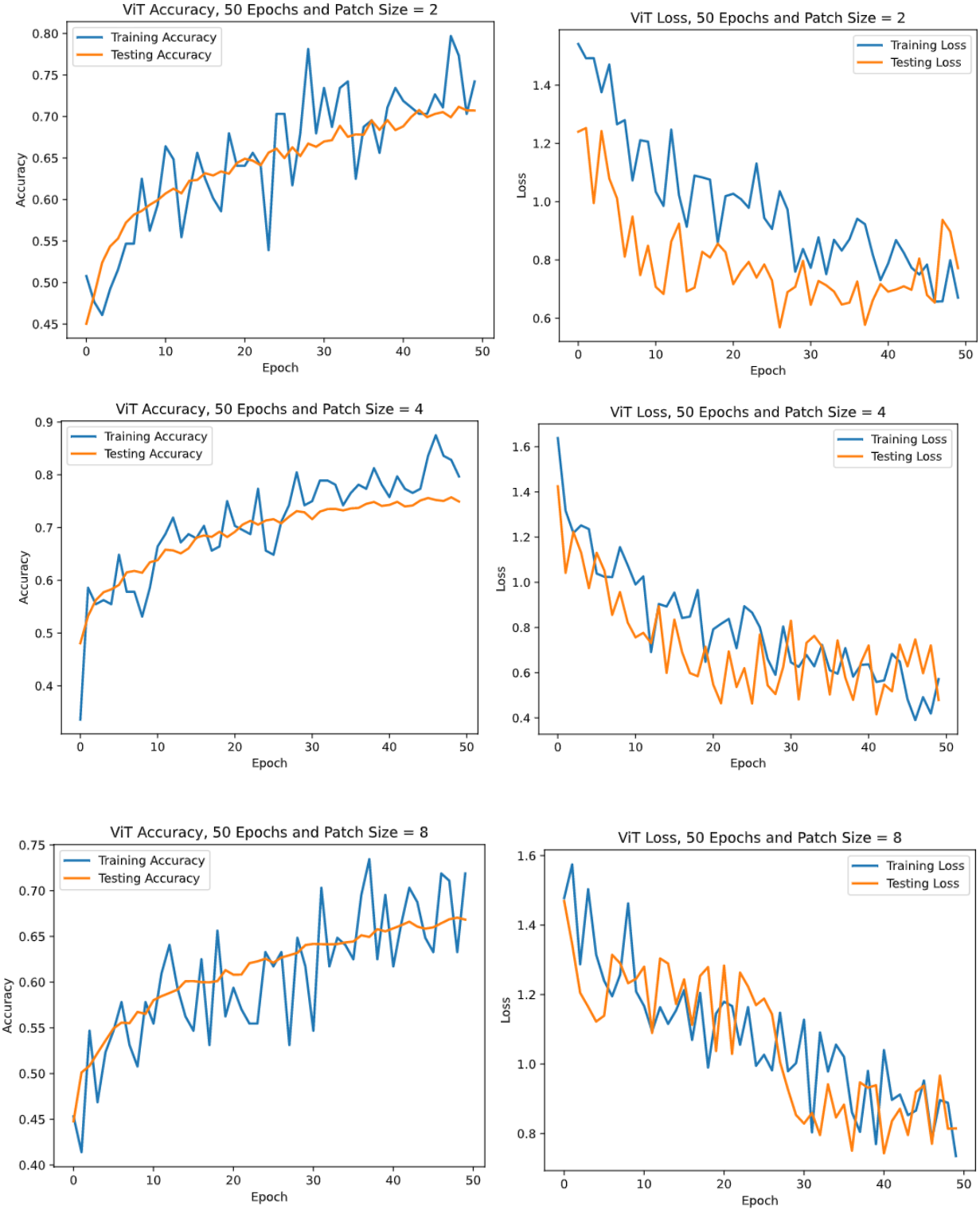
[9] Tutorial 15 (JAX): Vision Transformers. https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/JAX/tutorial15/Vision_Transformer.html.

[10] Assignment #4: Deep learning (due 04/07). https://www.seas.upenn.edu/~enm5310/assn4/.

[11] Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385.

Supplementary Information

Appendix A



Plots depicting the training and testing accuracy and training and testing loss for the three trials using the Vision Transformer model. By varying the patch size, we see there is a difference in model performance.