End-to-End Whole Slide Image Classification and Search using Set Representations

by

Bo Yang You

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Statistics

Waterloo, Ontario, Canada, 2022

Abstract

Due to the sheer size of gigapixel whole slide images (WSIs), it is difficult to apply deep learning and computer vision algorithms for digital pathology. Traditional approaches rely on extracting meaningful patches from a WSI and obtaining a representation for each patch individually. This approach fails to incorporate inherent information between the set of extracted patches. In this thesis, we approach the problem of WSI representation by using Set Transformers, a neural network architecture capable of incorporating the element-wise interactions of sets to obtain one global representation. We show through extensive experiments the representation capabilities of our method by outperforming existing patch-based approaches on search and classification tasks.

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Ali Ghodsi, for his continued support and feedback throughout my research journey. Thank you for welcoming me to the team and for giving me the freedom to pursue my research interests.

Thank you to Professor Mu Zhu and Professor Yeying Zhu for agreeing to be a part of the committee and for taking the time to review this thesis.

Many thanks to the members of the Data Analytics Lab, for their insightful presentations and research discussions. I am incredibly grateful for the collaborative learning experience especially during the period of remote learning.

To my parents and brother, thank you for your continued support and love.

Finally, I would like to thank Lillian Li, for being the best partner I could ever ask for.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning and neural networks have been effective at solving numerous problems in domains like computer vision and natural language processing. In the traditional deep learning framework, an input of fixed size in $\mathbb{R}^d$ is given to a neural network that learns to solve a specific problem. For non-conventional inputs like images or text, pre-processing is done to convert each image or text to this specific framework. Examples of such problems are object detection and translation. However, when the input is a set, where ordering of the elements and size of the set is arbitrary, the problem becomes much more complicated. In particular, the method has to be invariant to permutations of the input as well as being flexible to inputs of varying sizes. A case where set-valued functions are present is in the field of multiple instance learning where a set of instances are given as input and the target variable is a label for the entire set [21]. Additional problems involving set-valued inputs are sequence ordering and point cloud classification [21].

In this thesis, we focus on the problem of obtaining representations of whole slide images (WSIs). Whole slide imaging refers to the process of scanning glass slides in order to produce digitized slides. This process provides a use case for image processing techniques and computer vision algorithms to detect patterns in the underlying images. However, one of the major challenges in applying traditional computer vision algorithms and methods is the sheer size of a WSI. WSIs are extremely large files and the task of even loading an entire WSI into memory is infeasible. To address this issue, image patches are extracted and each processed individually [25, 14]. One disadvantage of this method is that information between patches are lost and the model is not invariant to permutations. By considering the entire WSI as a set and extracted patches as elements of the set, we frame the problem of WSI representation into a set representation problem.

Our contributions for the thesis are as follows. We provide an end-to-end process of WSI representation by using an attention based set-encoder called Set Transformer [21]. We are able to obtain a representation of a WSIs that can be used for tasks such as classification and search. We show through extensive experiments that this method outperforms on tasks compared to current state of the art.

In Chapter 2, we provide the relevant deep learning and WSI processing background. In Chapter 3 we discuss in detail the set-encoding framework and the inner workings of Set Transformer. Chapter 4 discuss our proposed model architecture as well as the dataset and data generation techniques used. We report our experiment results in Chapter 5.

Chapter 2

Background

2.1 Deep Learning

Definition 1 (Statistical Learning Problem)**.** Given i.i.d observations

$$\left\{(x_i, y_i) \in \mathbb{R}^m \times \mathbb{R}^d : 1 \leq i \leq n, i \in \mathbb{N}\right\}$$

from a distribution $\mathcal{D}$, we wish to find a function f in some hypothesis class $\mathcal{H}$ using $(x_i, y_i)_{i=1}^n$ such that the expected loss

$$\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

is minimized.

Deep Learning is a subclass of statistical learning where the class of functions $\mathcal{H}$ that we consider are neural networks. In recent years, deep learning algorithms have been very effective at learning representations of unstructured data such as images or text [20, 15, 10, 24, 27]. Advancements in computational resources as well as the availability of data have allowed deep learning algorithms to excel at a variety of tasks [27].

Definition 2. Let $l \in \mathbb{N}$ and $(h_1, \ldots, h_l) \in \mathbb{N}^l$. Let $W_i \in \mathbb{R}^{h_i \times h_{i+1}}$ be parameters for $1 \leq i \leq h_l$. Set $f_i(\boldsymbol{x}) = \sigma(W_i^T x + \boldsymbol{b}_i)$ where σ is an activation function and $\boldsymbol{b}_i$ is a bias term. We define a standard feed-forward neural network $f : \mathbb{R}^{h_1} \to \mathbb{R}^{h_l}$ by

$$f(\boldsymbol{x}) = f_l \circ \cdots \circ f_1(\boldsymbol{x})$$

We call functions f_i layers, l the network depth, h_i the number of nodes in layer i, W_i the weight matrix of layer i. f_1, f_l are the input and output layers, respectively while the remaining layers $f_2, \dots, f_{l-1}$ are called hidden layers. The weights W_i as well as the bias $\boldsymbol{b_i}$ are learnable parameters that are tuned based on an optimization procedure.

This class of functions is useful as it can be shown that the class of neural networks can approximate any continuous function [5, 16]. Thus if one can formulate a problem in the form of approximating a continuous function, then neural networks would be a good way to solve that problem.

The weights of the neural network are learned through a procedure called backpropagation using training data with respect to a loss function. Backpropagation [29] is a method of efficiently computing (sub)-gradients with respect to neuron weights in order to minimize the empirical loss. Since the introduction of the vanilla feed forward network (Definition 2), many new types of neural network architectures have been developed. We only focus on a few that are relevant to this thesis.

Convolutional Neural Networks (CNNs) are a type of neural network that is particularly useful for feature extraction, image classification, and image representation [20, 9]. The main difference between CNNs and regular neural networks is the use of convolution instead of general matrix multiplication in at least one of the layers.

Definition 3. The continuous convolution of f and g denoted by $\star$ is defined by

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

In the case of discrete inputs t, we define the discrete convolution by

$$(f \star g)(t) = \sum_{a=-\infty}^{\infty} f[a]g[t-a]$$

Additionally, we may also apply convolution over many dimensions. For example, in 2-dimensions, we have

$$(f \star g)(i,j) = \sum_{m}\sum_{n} f[m,n]g[i-m, j-n]$$

When the input is an (RGB) image, the input is usually a 3-dimensional tensor of size (h, w, c) where h, w, c denote the height, width, and number of channels, respectively.

In a convolutional layer, the input $X \in \mathbb{R}^{h \times w \times c}$ is first convolved with k filters of size (k_h, k_w, k_c) with a step-size of s. Then a non-linear activation function is applied to the generated feature map. Finally, there is a pooling layer which allows the network to generate a representation that is approximately invariant to translations of the input. By using kernels that are usually smaller in size than the input, CNNs typically have sparse interactions. Also, the use of a kernel allows parameter sharing of weights, enabling filters to learn meaningful features such as edges [9].

Pre-trained network architectures like **ResNet** [11], **DenseNet** [13], and **EfficientNet** [31] are all variants of CNNs with additional properties that have been previously trained on large image datasets. The learned kernels produce meaningful, rich features that are applicable to many downstream tasks and types of images. Using the pretrained kernels as a starting point, the network is then fine-tuned for a specific dataset or task.

ResNet [11] is a CNN type architecture that solve the problem of vanishing gradients and the performance degradation that comes with more convolutional layers. The difference in the architecture between residual blocks and convolutional blocks are the use of skip-connections.

Definition 4. Let X be the input and $\mathcal{F}(X, \{W_i\})$ denote any feature mapping layer with weight matrix W_i. Then a residual block is defined by

$$Y = \mathcal{F}(X, \{W_i\}) + X$$

where Y is the output of the layer.

The skip-connections allow the network to learn the identity function which ensures that more layers will perform at least as well as a lower number of layers.

DenseNet [13] is another CNN type architecture. It uses additional connections in order to improve information flow between layers. In particular, DenseNet propagates all features in a specific layer to all subsequent layers.

Definition 5. Let $\boldsymbol{x}_l$ denote the feature map from layer l, then a dense block is defined by

$$\boldsymbol{x}_l = \mathcal{F}_l(\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{l-1}\})$$

By propagating the feature map of previous layers, the size of the feature map would change from layer to layer. To address this, transition blocks are introduced which perform convolution and pooling to down-sample layers. Intuitively, the use of the feature maps in previous layers contribute to the "collective knowledge" of the network [13].

KimiaNet [28] is a patch-based model that is a fine-tuned version of DenseNet121. It has been shown to be effective at extracting features from individual patches of WSIs.

EfficientNet [31] addresses the issue of efficiently scaling a CNN to obtain optimal performance in terms of computational efficiency. The network considers the resolution or image size, the depth of the network, and the width of the network (the number of channels) as parameters to optimize. EfficientNet uses a coefficient ϕ that uniformly scales the depth, width, and resolution, denoted by d, w, r respectively. The scaling problem is defined by

$$\begin{aligned} d &= \alpha^{\phi} \\ w &= \beta^{\phi} \\ r &= \gamma^{\phi} \\ \text{s.t.} \alpha &\cdot \beta^2 \cdot \gamma^2 \approx 2 \\ \alpha &\geq 1, \beta \leq 1, \gamma \geq 1 \end{aligned}$$

The coefficient ϕ can be thought of as a variable that controls the amount of computational resources available for model scaling, while α, β, γ determine resource allocation.

Transformers [33] is a neural network architecture that relies the attention mechanism [23, 3]. Inspired by cognitive attention, the attention mechanism had seen great success in many NLP and computer vision tasks [15, 10, 23]. In particular, Transformer based architectures excel at learning representations of data so that the latent representations may be fine-tuned for downstream tasks. The attention mechanism can be thought of as a mapping between a set of query and key-value pairs to an output. The output is a weighted sum of the values where the weight of each value is a similarity measurement between the query and the corresponding key.

Definition 6. Let $Q \in \mathbb{R}^{n \times d_q}$ be a query matrix of n query vectors. Let $K \in \mathbb{R}^{n_v \times d_q}$, $V \in \mathbb{R}^{n_v \times d_v}$ be the key and value matrices respectively. Here, n_v denotes the number of value vectors and d_v denotes the dimension of the value vectors. The attention function

$$\text{Att} : \mathbb{R}^{n \times d_q} \times \mathbb{R}^{n_v \times d_q} \times \mathbb{R}^{n_v \times d_v}$$

is defined by

$$\text{Att}(Q, K, V) = \omega\big(QK^T\big)V$$

where ω is an activation function (typically Softmax).

We can see that the dot-product term $QK^T \in \mathbb{R}^{n \times n_v}$ is a measure of the similarity between query and keys. The activation function converts the similarity measurement to a weight so that the final output is a weighted sum of the value vectors. A value would get a higher weight if the corresponding key has a higher similarity with the query. There are variations of this function, with different ways to compute the weights. In the Transformers architecture, scaled dot-product attention is used. That is

$$\mathrm{Att}(Q, K, V) = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the keys. Additionally, rather than computing one attention function on Q, K, V, Transformers use multi-head attention.

Definition 7. Let Q, K, V be query, key, and value matrices respectively. Let $h \in \mathbb{R}^n$. For $i = 1, 2, \ldots, h$, let $W_i^Q \in \mathbb{R}^{n \times d_k}, W_i^K \in \mathbb{R}^{n \times d_k}, W_i^V \in \mathbb{R}^{n \times d_v}$ denote learnable linear projections for Q, K, V respectively. Let $W^O \in \mathbb{R}^{hd_v \times n}$. The multi-head attention operation is defined by

$$\mathrm{Multihead}(Q, K, V) = \mathrm{Concat}(o_1, \ldots, o_h)W^O$$

where

$$o_i = \mathrm{Att}(QW_i^Q, KW_i^K, VW_i^V)$$

In multi-head attention, the query, key, and value matrices are first linearly projected h times. Attention is then computed on the projections in parallel and the final result is concatenated together. This allows the model to attend to information from different subspaces[33].

2.2 Whole Slide Images

Due to the size of WSIs, existing methods first need to extract patches or a "mosaic" from the WSI in order to be able to apply existing computer vision techniques. The Yottixel algorithm [14] is an approach that has been shown to be effective at extracting meaningful patches from WSIs. We provide a brief overview of the Yottixel algorithm [14]:

1. Yottixel receives as input a WSI and begins by separating the tissue areas from the background. This is so that the (white) background which does not contain any tissue information is discarded.

2. The separated tissue area gets divided into a grid of patches at fixed magnification and size.

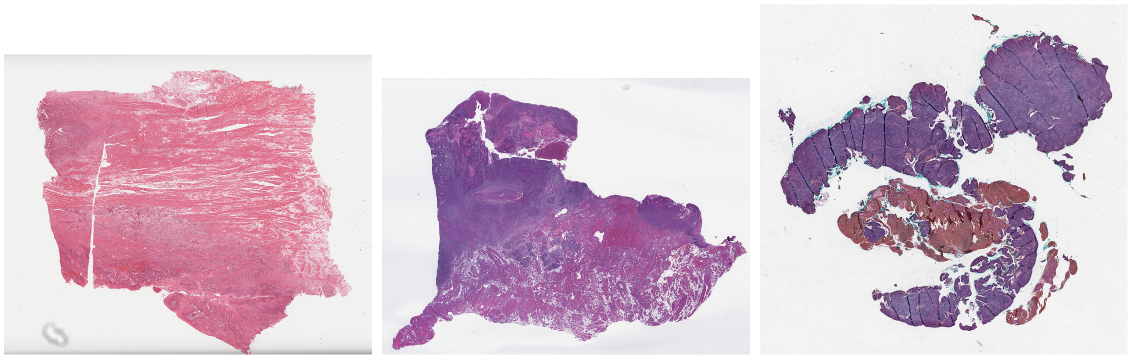
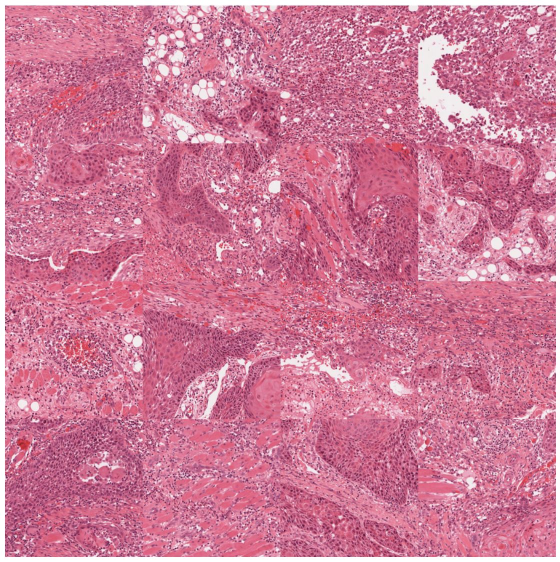

(a) Tongue (b) Mouth (c) Cervical

Figure 2.1: Examples of WSI thumbnails

3. Each patch is then clustered using k-means clustering in order to obtain groups of similar patches based on color composition. The number of clusters selected was based on manual observation of nine different types of visually distinct regions. The authors note that increasing the number of clusters may capture additional variability but do not seem to have relevance from a histopathology point of view.

4. A percentage of patches are then sampled (5% - 15%) from each cluster forming the mosaic of patches.

The patch extraction used in KimiaNet [28] is a slight modification of the original Yottixel algorithm. The intuition is that regions of interest such as high-grade carcinomas have high cellularity levels. Cellularity is measured by first deconvolving the patch color from RGB to hematoxylin and eosin channels using color deconvolution [28]. Then the patches are filtered based on the top 20% based on cellularity ratios. This approach filters the patches to ones that are more likely to be cancerous resulting in a condensed set of patches. As an example, Figure 2.1 contains thumbnails of entire WSIs and Figure 2.2 contains the extracted patches.


(a) Tongue

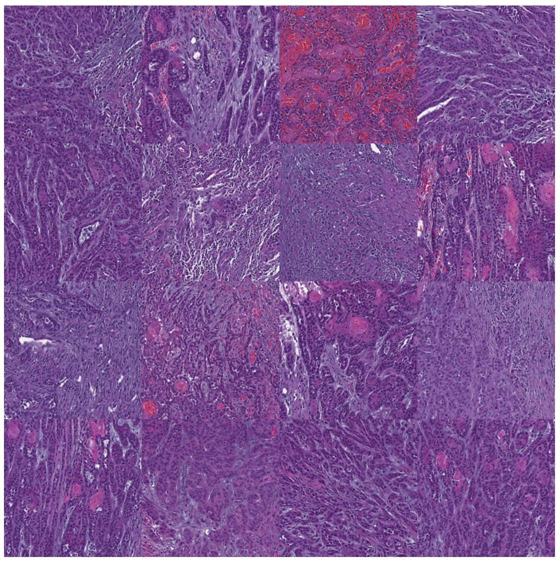

(b) Mouth

Figure 2.2: Examples of extracted patches from WSI using the Yottixel algorithm

Chapter 3

Set Encoding

Traditional machine learning algorithms work well when the input data is a fixed dimension d. However, the generalization to set-valued inputs is not trivial. Problems like point cloud classification, set anomaly detection and multiple instance learning all require the model to be invariant to permutation [21, 35]. In this section, we present the set classification problem, the structure of set-valued functions, and deep learning based methods used to generate representations of sets.

3.1 Set Classification

Definition 8. Let $\mathcal{X}$ be a set and $\mathcal{P}(\mathcal{X})$ denote the power set of $\mathcal{X}$. Suppose we have data instances $X_1, \dots, X_n \in \mathcal{P}(\mathcal{X})$, $y_1, \dots, y_n \in \{0, 1\}$. The set classification problem is to find a function $f : \mathcal{P}(\mathcal{X}) \to \{0, 1\}$ such that

$$f(X_i) = y_i$$

for $i = 1, \dots, n$. In practice, we often want to find an approximation $\hat{f}$ such that some loss function ℓ is minimized. That is,

$$\hat{f} = \min_{f \in \mathcal{F}} \ell(f(X), y)$$

In the deep learning setting, we restrict the class of functions $\mathcal{F}$ to be neural networks. Because the input data is a set, there are some additional properties that the function of interest should have. In particular, the function should be invariant to the ordering of the elements in the input set.

Definition 9. Let $\mathcal{X}$ be a set and let π be any permutation. We say that $f : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is permutation invariant if

$$f(\{x_1, \ldots, x_n\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(n)}\})$$

In the supervised learning setting, we also want the function to be invariant to the order of the training examples.

Definition 10. Let π be any permutation and suppose there are m training examples $(X_i, y_i)_{i=1}^m$. We say a function $\boldsymbol{f} : X^m \to y^m$ is permutation equivariant if

$$\boldsymbol{f}([X_1, \ldots, X_m]) = \left[f_{\pi(1)}(X_1), \ldots, f_{\pi(m)}(X_m)\right]$$

In the context of supervised learning and neural networks in particular, this property is important as it allows for training and performing inference in batches. Since compositions of permutation equivariant functions are permutation equivariant, if we are able to find a neural network layer that is permutation equivariant, we can compose those layers to generate a final model that is permutation equivariant.

3.2 Structure of Set-Valued Functions

Theorem 1 (Countable Case). *Let $\mathcal{X}$ be countable. A function $f : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is permutation invariant if and only if it can be decomposed in the form*

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right) \tag{3.1}$$

We say that a function is sum-decomposable if it satisfies Equation 3.1.

Proof. We provide a proof adapted from [35]. It is easy to see that by definition of 3.1 that f is permutation invariant as the summation function is permutation invariant. To show the reverse direction, notice that if we can choose ϕ so that $\Phi(X) = \sum_{x \in X} \phi(x)$ is one-to-one, we can choose $\rho = f \circ \Phi^{-1}$ so that

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right)$$

Since $\mathcal{X}$ is countable, there is a one-to-one mapping $c : \mathcal{X} \to \mathbb{N}$. Let $\phi(x) = 2^{-c(x)}$, so that each input x is a real number in base 2. Then $\Phi(X) = \sum_{x \in X} 2^{-c(x)}$ is a unique real number so that Φ is one-to-one as required. $\square$

Theorem 2 (Uncountable Case)**.** *Let $f : [0,1]^M \to \mathbb{R}$. f is permutation invariant if and only if it can be decomposed in the form*

$$f(x_1, \ldots, x_M) = \rho\left(\sum_{i=1}^{M} \phi(x_i)\right)$$

for a continuous outer function $\rho : \mathbb{R}^{M+1} \to \mathbb{R}$ and a continuous inner function $\phi : \mathbb{R} \to \mathbb{R}^{m+1}$.

Proof. The proof is much more involved than the countable case and is provided in Appendix A of [35]. □

The above theorems provide some characterizations of set-valued functions. In the countable case, the above theorem says that we only need a 1-dimensional latent space to represent a set. In the uncountable case, for a set-valued function of fixed size, the minimum size required to represent a set of size M is $M + 1$.

Additional results have been presented in [34] which dive deeper into the structure of set-valued functions. The above results consider general set-valued functions. However, in the supervised learning and deep learning framework, the classes of functions that are of interest are usually continuous. The following results deal with **continuous** set-valued functions.

Theorem 3 (Fixed Size)**.** *Let $f : \mathbb{R}^M \to \mathbb{R}$ be continuous. Then f is permutation invariant if and only if it is continuously sum-decomposable via $\mathbb{R}^M$.*

Theorem 4 (Variable Size)**.** *Let $f : \mathbb{R}^{\leq M} \to \mathbb{R}$ be continuous. Then f is permutation invariant if and only if it is continuously sum-decomposable via $\mathbb{R}^M$.*

The proofs of the above theorems are in [34]. This theorem says that for continuous set-valued functions, the size of the latent space needs to have a dimension of at least M. Note that this does not mean that all functions require a latent dimension size of at least M. However in the general case of continuous set-valued functions, we need a latent dimension size of at least M.

Although the latent dimension size of M can represent a set-valued function, learning this function using neural networks or other supervised learning methods is quite challenging. In the neural network framework, this challenge is greatly amplified due to the fact that the networks are composed of many permutation equivariant layers, thus making ϕ extremely complex. Because of this, we typically require a latent dimension that is much larger than M in order to learn a set-valued function well.

Looking at the structure of Equation 3.1, we see that the function consists of a few pieces. First, a mapping ϕ maps (encodes) each element from the set to a set of features. The features are then aggregated by summation (which is permutation invariant). Finally, another mapping ρ maps (decodes) the aggregated feature set to a final output. This process of encoding, aggregation, and then decoding is present in deep learning methods for set encoding.

3.3 Deep Sets

Deep Sets [35] is a deep learning framework for learning set-valued functions. Other architectures that learn set-valued functions such as PointNet [26] or T-Net [19] can be thought of as a specific instance of this framework. There are two major components to this framework. The first is a process to learn permutation invariant functions. By using the sum-decomposable characterization of set-valued functions in Equation 3.1, we note that we can replace ϕ, ρ by universal approximators like neural networks [5, 16]. Using neural networks to approximate ϕ, ρ, we obtain the following procedure:

1. Transform each element x_m into a representation $\phi(x_m)$.
2. Aggregate each element representation $\phi(x_m)$, say z.
3. Transform z into a representation of the entire set $\rho(z)$.

By backpropagation, we are then able to learn an approximation of ϕ, ρ using neural networks. The second component to this framework is to create neural network layers that are permutation equivariant.

Lemma 1. *Let f_Θ be a standard neural network layer, i.e.*

$$f_\Theta(x) = \sigma(\Theta x)$$

where $\Theta \in \mathbb{R}^{M \times M}$ is the weight matrix and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function. f_Θ is permutation equivariant if and only if

$$\Theta = \lambda I + \gamma(\mathbf{1}\mathbf{1}^T)$$

for $\lambda, \gamma \in \mathbb{R}$, $\mathbf{1} = [1, \ldots, 1]^T$ and $I \in \mathbb{R}^{M \times M}$ is the identity matrix.

The result above [35] can be generalized to higher dimensions when λ, γ are matrices. Furthermore, it is possible to change the elements in the layer to obtain more complex permutation equivariant layers. By stacking these permutation equivariant layers, it is possible to build a more complex Deep Sets model.

3.4 Set Transformer

Although Deep Sets is a framework that can learn set-valued functions, the simple sum aggregation step after encoding each element fails to model interactions between elements of the set. Set Transformer [21] is a set-valued generalization of the original Transformer architecture in [33]. It uses a self-attention mechanism in order to encode pairwise and higher order interactions between elements of a set. Like Deep Sets, Set Transformer consists of an encoder, decoder and a new aggregation function that leverages self-attention.

Definition 11. Let $X, Y \in \mathbb{R}^{n \times d}$ represent two sets of d-dimensional vectors. The Multihead Attention Block (MAB) parameterized by ω is given by

$$\text{MAB}(X, Y) = \text{LN}(H + f_\Theta(H))$$

where $H = \text{LN}(X + \text{Multihead}(X, Y, Y; \omega))$, f_Θ is a row-wise feed forward neural network layer, and LN is a layer normalization layer [2]. Note that the MAB layer is very similar to the encoder of the Transformer architecture [33] without the use of positional encoding and dropout layers.

Definition 12. Let $X \in \mathbb{R}^{n \times d}$ be a set of n d-dimensional vectors. The Set Attention Block (SAB) is given by

$$\text{SAB} = \text{MAB}(X, X)$$

Notice that SAB takes in a set as input and outputs a set of equal size. Additionally, it performs self attention between elements of the set resulting in pairwise interactions. Additionally, composing layers of SAB will result in higher order interactions between elements of the set. One disadvantage of this architecture is the computational complexity. Computing self-attention between all elements in the set has quadratic time complexity $O(n^2)$ which may become costly for large sets. To address this, m learnable inducing vectors $I \in \mathbb{R}^{m \times d}$ are introduced.

Definition 13. Let $I \in \mathbb{R}^{m \times d}$ be m d-dimensional learnable inducing points. The Induced Set Attention Block (ISAB) is given by

$$\text{ISAB}_m(X) = \text{MAB}(X, H) \in \mathbb{R}^{n \times d}$$

where $H = \text{MAB}(I, X) \in \mathbb{R}^{m \times d}$.

The ISAB first attends I with X to produce an output that contains information about X. Then that output is attended again with X to produce the final output. This is similar to an autoencoder where we first project X do a lower dimensional latent space and then reconstruct to produce an output. Notice that the computational complexity is now $O(nm)$ so for smaller m, this operation is approximately linear. Furthermore, it can be shown that the SAB and ISAB functions are permutation equivariant [21] which makes it possible to compose these layers to learn more complex functions.

In the Deep Sets framework, the aggregation function used was summation. Other functions like max, min and mean have also been used. Set Transformer uses a pooling function that relies on multihead attention.

Definition 14. Let $S \in \mathbb{R}^{k \times d}$ denote k d-dimensional seed vectors. Let $Z \in \mathbb{R}^{n \times d}$ denote the set of features generated from the encoder. The Pooling by Multihead Attention (PMA) layer is given by

$$\text{PMA}_k(Z) = \text{MAB}(S, f_\Theta(Z))$$

where f_Θ is a feed-forward neural network layer.

Typically, $k = 1$ is used for most scenarios. The pooling by attention operation is essentially a weighted average. Intuitively, this makes sense since the weight of each element of a set should not necessarily be equal when predicting a target. Additionally this is a generalization of the summation function used in Deep Sets. Notice that if the key and query values are $\mathbf{1} = (1, \ldots, 1)^T$, we obtain the regular summation function when computing attention. Set Transformer performs a weighted average based on the attention values of each element in a set rather than assigning each element equal weight.

To frame this architecture like Equation 3.1, we have that

$$\phi(X) = f_1 \circ f_2 \circ \cdots \circ f_l(X) = Z \in \mathbb{R}^{n \times d}$$

where f_i is an ISAB or MAB block. For the decoder layer, Set Transformer applies a feed forward layer and SAB layer to the output of the pooling layer. That is,

$$\rho(Z) = f_\Theta \circ \text{SAB}(Z)$$

Putting this all together, we have that

$$f(X) = \rho(\text{PMA}(\phi(X)))$$

Theorem 5. *The Set Transformer architecture is permutation invariant.*

Proof. We provide an outline of the proof. Notice that SAB and ISAB are permutation equivariant, so the composition of these blocks will be permutation equivariant. This means that the encoder that maps an input X to a latent representation Z is permutation equivariant. Furthermore, we have that the PMA layer in the decoder is permutation invariant. Use a similar argument as in Theorem 2 to show that Set Transformer is permutation invariant. □

Theorem 6. *The Set Transformer architecture is a universal approximator of set-valued functions.*

A proof of the above theorem is provided in the supplementary materials of [21]. These results show the promise of using Set Transformers for approximating set-valued functions. Additionally, the inherent properties of using self-attention allows Set Transformer to capture the pairwise and higher order interactions between elements of the set, potentially leading to better representations.

Chapter 4

Method

4.1 WSI Representation in the Set Encoding Framework

As WSI images are extremely large, training a model on the entire WSI is infeasible. Additionally, only a diagnosis label is provided for the entire WSI. However, we do not know where the cancerous regions within a WSI are. Because of this, existing works focus on processing patches from the WSI.

Putting this together within the context of set encoding, we treat each WSI as a set, and the patches within the WSI as elements of the set. Formally, we let $\mathcal{W}$ be the space of all WSIs. Fix n to be the number of patches for each WSI and d to be the dimension of each patch. Let Y be the space of labels. The goal is to learn a set-valued mapping $f : \mathcal{W} \to Y$ such that

$$f(W) = y$$

for each sample $(W, y) \in \mathcal{W} \times Y$.

Note the use of a fixed set size n. Thus for WSIs with more than n patches we select randomly a subset of size n. For WSIs with m patches for $m < n$, we pad the remaining $m - n$ patches with $\mathbf{0} \in \mathbb{R}^d$.

4.2 Dataset

The main dataset used for the experiments was obtained from The Cancer Genome Atlas (TCGA). TCGA is a publicly available, rich dataset containing WSIs of over 20,000 primary cancer and matched normal samples spanning over 33 cancer types. We used a subset of the TCGA data containing 8,544 WSIs. Each WSI belonged to 1 of 13 different tumor types and 1 of 30 different subtypes. to be consistent with the dataset used in [28]. A full breakdown of the class label and the corresponding number of slides is presented in Table 4.1. Of the 8,544 WSIs, 7,097 was used for training, 703 was used for validation, and 744 was used for testing resulting in an approximate 80-10-10 split. We process each WSI using the modified Yottixel algorithm described in Section 2 to obtain a set of 242,202 patches, 24,646 patches, and 116,088 patches from the training, validation, and testing datasets, respectively. Table 4.1 shows the breakdown of the number of WSIs in each of the datasets and classes.

4.2.1 Data Generation

As described above, our method requires the use of a fixed set size n. However, many images in the training dataset contains more than n patches. Thus in order to create a richer dataset for the model, we can generate data by selecting subsets of patches from each WSI. To generate the dataset, we fix n and s which denote the number of patches and number of samples per WSI, respectively. For each WSI, we generate a subset of size n from $\{0, \ldots, w-1\}$ where w denotes the number of patches in that WSI. We repeat this procedure s times to obtain the new training dataset. Additionally, we apply relevant transformations such as normalization, horizontal and vertical flips, and rotations so that the network does not learn to memorize training instances and to improve the generalization capabilities of our model. We also resize the patches from a dimension of $(1000, 1000, 3)$ to dimension $(224, 224, 3)$ so that the patches can be fed into the backbone networks.

4.3 Model Architecture

Our proposed model consists of three main stages. We first extract features from each patch using a CNN based backbone. Then we obtain a permutation invariant embedding of the set of features using Set Transformer. Finally, we apply a classification network in order to predict the label. We choose Set Transformer over other set encoders like Deep

Tumor Site	**Subtype**	**Train**	**Validation**	**Test**
Brain	LGG	574	40	39
Brain	GBM	750	36	35
Endocrine	THCA	406	51	51
Endocrine	ACC	210	5	6
Endocrine	PCPG	130	15	15
Gastro.	ESCA	99	14	14
Gastro.	COAD	236	31	32
Gastro.	STAD	196	27	30
Gastro.	READ	89	9	12
Gynaeco.	UCS	35	3	3
Gynaeco.	CESC	164	22	17
Gynaeco.	OV	78	10	10
Liver, panc.	CHOL	26	4	4
Liver, panc.	LIHC	284	35	35
Liver, panc.	PAAD	106	12	12
Melanocytic	SKCM	206	25	24
Melanocytic	UVM	24	4	4
Prostate/testis	PRAD	361	38	40
Prostate/testis	TGCT	156	13	13
Pulmonary	LUAD	303	38	38
Pulmonary	LUSC	361	41	43
Pulmonary	MESO	50	5	5
Urinary tract	BLCA	308	34	34
Urinary tract	KIRC	404	50	50
Urinary tract	KIRP	239	25	28
Urinary tract	KICH	97	11	11
Breast	BRCA	797	87	91
Head, neck	HNSC	246	32	32
Sarcoma	SARC	134	13	13
Thymoma	THYM	28	3	3

Table 4.1: Number of WSIs by dataset

Sets [12] due to the attention-based mechanism of Set Transformer. This allows the model to incorporate patch-level interactions which a Deep Sets architecture lacks.

4.3.1 Feature Extraction

The first component of the network is to extract meaningful features from each image patch. This step mimics the role of ϕ in Definition 3.1. Rather than training a model from scratch, we use CNN based backbones trained on ImageNet [6]. In particular, we ran our experiments with three sets of backbones: ResNet, DenseNet, and EfficientNet.

Let W be WSI image of n patches, b be the batch size, and ϕ be the backbone network. Thus our input tensor to the network has dimensions $(b, n, 224, 224, 3)$. Prior to feeding W to the network, we apply a reshape transformation $g : \mathbb{R}^{b\times n\times 224\times 224\times 3} \to \mathbb{R}^{(b\cdot n)\times 224\times 224\times 3}$ to obtain a tensor W' of dimension $(b \cdot n, 224, 224, 3)$. This is so that feature extraction is performed on the patch level and also utilizes the parallelism nature of deep learning packages [12]. The size of the features extracted d vary with $d = 2048$ for ResNet, $d = 1024$ for DenseNet, and $d = 1280$ for EfficientNet. We then reshape W' to obtain a tensor of dimension (b, n, d). This can be thought of applying the inverse $g^{-1} : \mathbb{R}^{(b\cdot n)\times 224\times 224\times 3} \to \mathbb{R}^{b\times n\times 224\times 224\times 3}$.

Putting this all together, we obtain a set of features Z for the set of WSI patches with

$$Z = g^{-1}(\phi(g(W))) \in \mathbb{R}^{b\times n\times d}$$

4.3.2 Set Encoding

The set encoding step mimics the role of ρ in Definition 3.1. At the beginning of this stage, we are given a set of features $Z \in \mathbb{R}^{b\times n\times d}$. We use the Set Transformer architecture described in Section 3 to encode the entire set of features into one representation. There are a few parameters of interest when constructing the Set Transformer architecture:

1. The number of ISAB blocks n_b.
2. The dimension of the inducing points m.
3. The number of attention heads h.
4. The dimension of the set representation n_e.

The use of the PMA layer ensures that the network is permutation invariant. The use of ISAB layers reduce the computational complexity of computing self-attention. Letting ρ be the Set Transformer, we obtain a representation of size n_e. Thus after the set encoding stage, we obtain

$$W^\star = \rho(Z) \in \mathbb{R}^{b \times n_e}$$

4.3.3 Classification

Notice that $W^\star$ is the final representation of the WSI. However, in order to train the entire network architecture, we use a classification network layer in order to predict the class labels. The final stage of the architecture is to use a classification layer that takes in as input a batch of WSI representations $W^\star \in \mathbb{R}^{b \times n_e}$, and produces an output $y \in \mathbb{R}^{b \times 30}$. Our choice of network for the classification stage is a 4-layer feed forward neural network. The hidden layers have sizes of $512, 256$ and 128 respectively. The input size is n_e and the output size is a vector of logits with size 30. As the focus of this work is on the set representations of the WSI itself, we did not tune the hyper-parameters of the classification architecture.

We use the cross-entropy loss function with label smoothing in order to prevent overconfident predictions. The cross entropy function is defined by

Definition 15. Let n be the number of samples and C be the number of classes. Let $(x_i, y_i)_{i=1}^n$ with x_i denoting a vector of logits and y_i being a one-hot encoded vector of the true class label. The cross-entropy loss is defined by

$$\ell(\mathbf{x}, \mathbf{y}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_{i,j} \log \frac{\exp(x_{i,c})}{\sum_{k=1}^{C} \exp(x_{i,k})}$$

with $y_{i,j}$ denoting the j-th element of the one-hot encoded label of sample i and $x_{i,k}$ denoting the k-th element of the logits of sample i.

In label smoothing, we fix a smoothing parameter α and replace y with a mixture of y with a uniform distribution. That is,

$$y_i' = (1-\alpha) y_i + \frac{\alpha}{C}$$

4.3.4 Improving Model Accuracy Using Game Theory

Throughout our experiments, we notice a significant gap between top-1 accuracy and top-k accuracy. This suggests that the model contains useful information about the task, but is not confident enough to predict accurately. Drawing inspiration from game theory, we address this problem by framing it as a Lewis signaling game [30, 9].

Formally, a Lewis signaling game consists of a set of states X, a sender f, a receiver g and a reward function R. The sender is a mapping $f : X \to M$ that maps a state to a specific message. The receiver is a mapping $g : X \times M \to Y$ that maps a message (along with the state) to a specific action. Finally, the reward is a function $R : S \times Y \to \mathbb{R}$ that is optimized.

In the context of our set classification problem, we have X being the space of WSIs, M being the space of representations, and Y being the specific labels.

We keep the sender and receiver architecture the same with the exception of an additional message $m \in M$ which is the representation generated by f. Additionally, we modify the loss function used to train our network. Let ℓ_α denote the cross-entropy loss with label smoothing parameter α. We define our training loss function ℓ as a weighted sum of the sender and receiver loss.

$$\ell(x) = \beta\ell_\alpha(x, f(x)) + (1-\beta)\ell_\alpha(x, g(x, f(x))), \beta \in (0, 1)$$

This loss allows the sender model to continuously improve the quality of messages being sent to the receiver model as well as allowing the receiver model to generate more accurate predictions using the representations. Similar to the mode collapse problem of GANs [9, 18], our proposed training method runs into a similar issue when the initial messages generated by f are not meaningful. To address this problem, we first train f by itself and then train f, g simultaneously when f is able to send meaningful messages to g. We report our results in Section 5.

Chapter 5

Experiments

In order to test our proposed architecture, we trained it using a number of different configurations. Additionally, we compare our model to KimiaNet [28] in both horizontal and vertical search tasks.

5.1 Training

In all cases, we trained our models for 20 epochs. The batch size varied depending on the GPUs that were available from Compute Canada (Graham Cluster). All models were trained on nodes with 2 NVIDIA P100 Pascal GPUs and we note that the use of distributed training methods via additional nodes and GPUs can improve the training times even more.

Figure 5.1 shows the distribution of patches per WSI across the various datasets. We fix the number of patches used as input to be 32. 32 was selected as it was the median patch size of all training and validation images. This resulted in a dataset with 50% of the training dataset that required padding and 50% of the dataset that required sampling. Notice that this also allows the model to generalize to WSIs with less than 32 patches. We also note that the testing dataset had a median patch size of 146.

5.2 Architecture Selection

We run a number of experiments to decide the final set of hyper-parameters used for our network. In particular, we wish to decide the CNN backbone architecture, and the

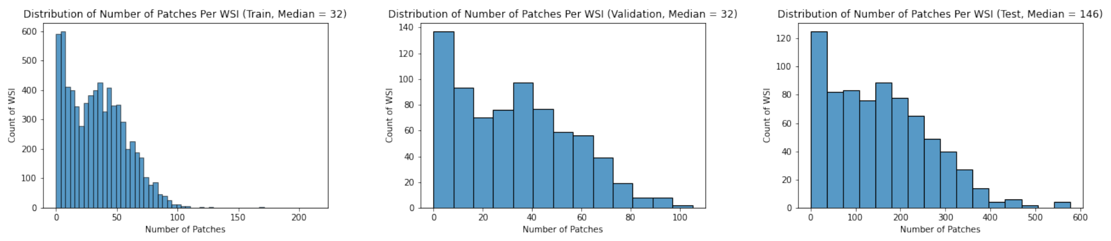


Figure 5.1: Distribution of patches per WSI across the training, validation, and testing datasets

parameters used for Set Transformer. For each set of parameters in Set Transformer, we run three experiments with each of the candidate backbone architectures. We use a smaller generated dataset in order to quickly evaluate the performance of each set of parameter configurations and then train a final model on a larger generated dataset. For each WSI, we sample 32 patches 3 times. This results in a total of 21,378 WSIs for the training set. We train each candidate model for a total of 10 epochs and select the model based on the lowest validation loss.

We use the Optuna [1] package for hyperparameter tuning. Each architecture was tuned for a total of 10 trials and we prune trials based on the median stopping rule. That is, we stop a trial early if the trial's best intermediate result is worse than the median of the intermediate results of previous trials at the same step. Using the method discussed in Section 4.3.4, we first train a sender model for 10 epochs. Then we train both sender and receiver models simultaneously for another 10 epochs with early stopping. We also use a larger generated dataset which samples each WSI 8 times for a total of 57,008 training samples.

5.3 WSI Search

In addition to classification accuracy, we evaluate the effectiveness of our representations by search tasks. The search tasks refers to how well the algorithm can select similar WSIs from a database. From the point of view of a pathologist, we are given a query WSI Q that is completely new (no previous diagnosis). We wish to find similar cases in an existing database of WSIs so that the pathologist has previous data to compare to. An effective WSI representation should be able to find WSIs in the database that have similar diagnosis or subtypes. Notice that this task is different from classification. For classification, we only give a class label as output, however we do not know which WSIs are similar to our input. In search, we solely use the representation in order to find similar WSIs in the database.

There are two kinds of search tasks. **Horizontal search** refers to how well the algorithm can find the overall tumor type (brain, breast, liver, etc.). **Vertical search** refers to how well the representations can find the correct subtype (LGG, GBM, etc) given an overall tumor type. In both tasks, we compare our results with KimiaNet. Since KimiaNet does representations on a patch level, the method of searching is slightly different than comparing WSI representations directly. To compare the similarity between two WSIs, KimiaNet uses the median-of-min approach. That is, the minimum distance of each patch in the query WSI is first calculated when compared to all the patches of the other WSI. The median value of the minimum distances is then taken as the similarity between the query WSI and the candidate WSI. Thus the similarity measure is defined by

$$s(Q, W) = \text{Median}_{q \in Q} \left(\min_{w \in W} d(q, w) \right)$$

for patches $q \in Q$ and $w \in W$.

In order to keep the method similar for comparison, we obtain a representation for each patch by padding each WSI with zeros. We then apply the same median-of-min method with Euclidean distance in order to find the k-nearest neighbors to the query patch with $k = 3$.

We also report the results of using the representations of the WSI directly. However, from the point of view of a pathologist, this may not be as meaningful. Additionally, our method allows for representations for subsets of WSIs. One potential application of this is to obtain more refined search results. That is, given a specific subset of patches in a sub-region of the WSI, one may search for similar representations of subsets in the database. Comparing representations on a patch level is much more meaningful as it allows pathologists to isolate areas of interest.

5.3.1 Horizontal Search

We compare our results against KimiaNet. The metric for evaluations in the horizontal search task is classification accuracy. Table 5.1 shows the results of our method compared to KimiaNet. RN, DN, EN, KN correspond to the ResNet, DenseNet, EfficientNet, KimiaNet backbone architectures of our method, respectively. The best performing scores are highlighted in green.

Figure 5.2 also shows the average accuracy across all horizontal classes (along with the standard deviation). We observe comparable results to KimiaNet and outperforms in identifying several of the horizontal search classes.

Tumor Type	**KimiaNet**	**RN**	**DN**	**EN**	**KN**
Brain	**99**	97	96	97	97
Breast	**91**	**91**	85	88	89
Endocrine	92	92	81	**94**	92
Gastro.	**84**	78	69	73	72
Gynaec.	57	63	30	**67**	60
Head/neck	**88**	75	41	81	66
Liver	**88**	76	67	75	80
Melanocytic	**86**	68	43	71	61
Mesenchymal	69	85	**100**	85	92
Prostate/testis	**96**	87	87	94	92
Pulmonary	86	81	80	**90**	81
Urinary tract	89	93	88	90	**95**

Table 5.1: 3-nearest neighbors accuracy (%) for horizontal search task.

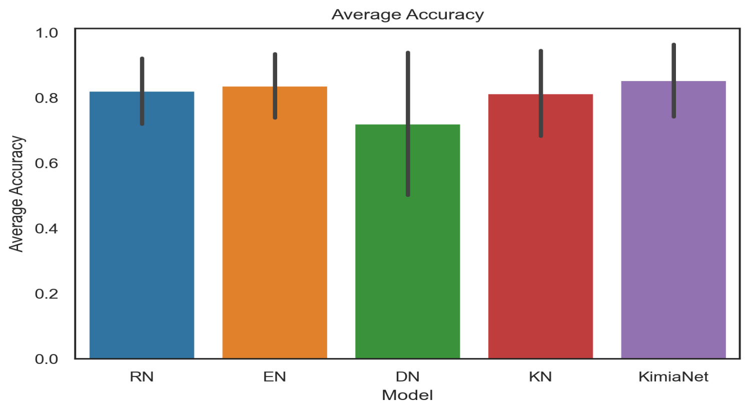


Figure 5.2: Average accuracy with standard deviation for horizontal search task.

5.3.2 Vertical Search

The metric for evaluations in the vertical search task is the F_1 score. Table 5.2 shows the results of our method compared to KimiaNet. RN, DN, EN, KN correspond to the ResNet, DenseNet, EfficientNet, KimiaNet backbone architectures of our method, respectively. The best performing scores are highlighted in green.

We see that our methods outperform KimiaNet overall in the vertical search tasks. Specifically, we obtain better results in the Brain, Endocrine, Gynaeco., Liver, panc., and Pulmonary sites and achieve comparable performance in the remaining categories.

Site	**Subtype**	**KimiaNet**	**RN**	**DN**	**EN**	**KN**
Brain	LGG	85	91	89	91	**95**
Brain	GBM	83	90	87	90	**94**
Endocrine	THCA	**100**	**100**	**100**	99	98
Endocrine	ACC	55	**92**	71	80	**92**
Endocrine	PCPG	85	**97**	86	93	**97**
Gastro.	ESCA	83	90	63	**92**	67
Gastro.	COAD	76	**85**	66	79	**85**
Gastro.	STAD	**86**	84	70	82	83
Gastro.	READ	30	59	29	56	**63**
Gynaeco.	UCS	86	**100**	75	75	75
Gynaeco.	CESC	**97**	**97**	91	94	91
Gynaeco.	OV	**95**	**95**	82	89	**95**
Liver, panc.	CHOL	40	**89**	67	40	0
Liver, panc.	LIHC	97	**99**	**99**	97	93
Liver, panc.	PAAD	82	91	**96**	**96**	89
Melanocytic	SKCM	**98**	96	92	94	96
Melanocytic	UVM	**86**	67	0	40	67
Prostate/testis	PRAD	**100**	**100**	99	**100**	99
Prostate/testis	TGCT	**100**	**100**	96	**100**	96
Pulmonary	LUAD	78	85	**90**	**90**	88
Pulmonary	LUSC	84	89	89	**91**	**91**
Pulmonary	MESO	75	**89**	33	**89**	**89**
Urinary tract	BLCA	**96**	94	89	94	93
Urinary tract	KIRC	**99**	94	95	97	96
Urinary tract	KIRP	91	90	91	**96**	93
Urinary tract	KICH	**86**	**86**	80	84	84

Table 5.2: 3-nearest neighbors F_1 score (%) for vertical search task.

Chapter 6

Conclusion

We conclude the thesis by summarizing our contributions and suggesting directions for future research. We developed an end-to-end approach for representation of WSIs that is flexible to number of extracted patches and obtains comparable results for classification and search tasks. The extracted representation is permutation invariant and is easily adaptable to any feature extractor on the patch level. By using the Set Transformer architecture, we are able to incorporate higher order patch level interactions. We show that our representations show success in both classification and search tasks. In particular, we obtain comparable results with KimiaNet on horizontal search tasks and outperform on vertical search tasks.

There are a few possible directions for future research. As discussed above, our method is adaptable to use any feature extractor backbone. Newer methods of Vision Transformers like ViT [8] and Swin Transformers [22] may serve as better feature extractors of patches. The downside to using these methods is the additional computational complexity it adds to the entire model.

Another direction of research is adapting Set Transformer to be flexible to WSIs with an arbitrary number of patches. Currently, our method fixes a patch size n and is flexible to WSIs with a number of patches that is smaller than n. However, by using an additional aggregation step, it may be possible to obtain a set valued model that is flexible to any number of patches [4].

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework, July 2019. Number: arXiv:1907.10902 arXiv:1907.10902 [cs, stat].

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. arXiv: 1607.06450.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. arXiv: 1409.0473.

[4] Andreis Bruno, Jeffrey Ryan Willette, Juho Lee, and Sung Ju Hwang. Mini-Batch Consistent Slot Set Encoder for Scalable Set Encoding. page 16.

[5] G Cybenkot. Approximation by superpositions of a sigmoidal function. page 12.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. page 8.

[7] Neofytos Dimitriou, Ognjen Arandjelović, and Peter D. Caie. Deep Learning for Whole Slide Image Analysis: An Overview. *Frontiers in Medicine*, 6, 2019.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, June 2021. arXiv: 2010.11929.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R. Martin, Ming-Ming Cheng, and Shi-Min Hu. Attention Mechanisms in Computer Vision: A Survey. *arXiv:2111.07624 [cs]*, November 2021. arXiv: 2111.07624.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[12] Sobhan Hemati, Shivam Kalra, Cameron Meaney, Morteza Babaie, Ali Ghodsi, and Hamid Tizhoosh. CNN and Deep Sets for End-to-End Whole Slide Image Representation Learning. In *Proceedings of the Fourth Conference on Medical Imaging with Deep Learning*, pages 301–311. PMLR, August 2021. ISSN: 2640-3498.

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, January 2018. arXiv: 1608.06993.

[14] S. Kalra, C. Choi, S. Shah, L. Pantanowitz, and H. R. Tizhoosh. Yottixel – An Image Search Engine for Large Archives of Histopathology Whole Slide Images. *arXiv:1911.08748 [cs, eess]*, November 2019. arXiv: 1911.08748.

[15] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Computing Surveys*, page 3505244, January 2022. arXiv: 2101.01169.

[16] Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. page 22.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.

[18] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On Convergence and Stability of GANs, December 2017. arXiv:1705.07215 [cs].

[19] Jean Kossaifi, Adrian Bulat, Georgios Tzimiropoulos, and Maja Pantic. T-Net: Parametrizing Fully Convolutional Nets with a Single High-Order Tensor. *arXiv:1904.02698 [cs]*, April 2019. arXiv: 1904.02698.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[21] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3744–3753. PMLR, May 2019. ISSN: 2640-3498.

[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. *arXiv:2103.14030 [cs]*, August 2021. arXiv: 2103.14030.

[23] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv:1508.04025 [cs]*, September 2015. arXiv: 1508.04025.

[24] Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco-Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep Learning vs. Traditional Computer Vision. *arXiv:1910.13796 [cs]*, 943, 2020. arXiv: 1910.13796.

[25] Liron Pantanowitz, Paul N. Valenstein, Andrew J. Evans, Keith J. Kaplan, John D. Pfeifer, David C. Wilbur, Laura C. Collins, and Terence J. Colgan. Review of the current state of whole slide imaging in pathology. *Journal of Pathology Informatics*, 2:36, August 2011.

[26] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv:1612.00593 [cs]*, April 2017. arXiv: 1612.00593.

[27] Maithra Raghu and Eric Schmidt. A Survey of Deep Learning for Scientific Discovery. *arXiv:2003.11755 [cs, stat]*, March 2020. arXiv: 2003.11755.

[28] Abtin Riasatian, Morteza Babaie, Danial Maleki, Shivam Kalra, Mojtaba Valipour, Sobhan Hemati, Manit Zaveri, Amir Safarpoor, Sobhan Shafiei, Mehdi Afshari, Maral Rasoolijaberi, Milad Sikaroudi, Mohd Adnan, Sultaan Shah, Charles Choi, Savvas Damaskinos, Clinton JV Campbell, Phedias Diamandis, Liron Pantanowitz, Hany Kashani, Ali Ghodsi, and H. R. Tizhoosh. Fine-Tuning and Training of DenseNet for Histopathology Image Representation Using TCGA Diagnostic Slides. *arXiv:2101.07903 [eess]*, January 2021. arXiv: 2101.07903.

[29] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. page 4, 1986.

[30] Brian Skyrms. The flow of information in signaling games. *Philosophical Studies*, 147(1):155–165, January 2010.

[31] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, September 2020. arXiv: 1905.11946.

[32] Hamid Reza Tizhoosh and Liron Pantanowitz. Artificial Intelligence and Digital Pathology: Challenges and Opportunities. *Journal of Pathology Informatics*, 9:38, November 2018.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv: 1706.03762.

[34] Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the Limitations of Representing Functions on Sets. *arXiv:1901.09006 [cs, stat]*, October 2019. arXiv: 1901.09006.

[35] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep Sets. *arXiv:1703.06114 [cs, stat]*, April 2018. arXiv: 1703.06114.