Determining Cancer Hormone Receptor Status from H&E Stained Tissue Using Deep Learning

Capstone Report

By: Dean Kelley
deanak@uw.edu

Committee Members:
Dr. Anderson Nascimento, Chair
Dr. Martine De Cock
Dr. Juhua Hu
University of Washington, Tacoma
Winter 2024

Abstract: Image classification using deep learning methods like convolutional neural networks (CNNs) is used by many today for a variety of applications, ranging from facial recognition to autonomous vehicles and object detection in computer vision systems. One area that has seen significant growth is classifying medical tissue sample images for faster prognosis and in turn, faster treatment decisions since every moment counts for those affected by disease. It is already well studied that specific types of cancer such as those which are Estrogen Receptor positive (ER+) can be classified to increasingly higher accuracy scores but many types require further research. A machine learning model and adequate data processing pipeline must be constructed to classify these cancer types.

This paper outlines the development of an open-source model and data processing pipeline. The model efficiently utilizes whole slide images (WSI) of hematoxylin and eosin (H&E) stained tissue samples, specifically those collected by the Cancer Genome Atlas Project (TCGA), to accurately infer the hormone receptor statuses (HRS) associated with cancer. We have successfully built a model, which we have named ReceptorNeXt, that implements a newer and more effective feature extractor capable of identifying HR indicators to predict HRS. Our model is trained using WSI-level annotations using data from 1088 patients, and performs well with an area under the curve (AUC) of 0.79, 0.73, 0.65, and 0.73 for estrogen, progesterone, HER2, and triple-negative respectively. To enhance efficiency and effectiveness, we utilize various techniques such as multiprocessing, automatic mixed precision (AMP), squeeze and excitation (SE), and ensembles.

1 Introduction

Cancer has been affecting humankind for millennia, with records dating back to 3000 BCE. It is one of the leading causes of death today and, while we have learned so much about what causes it, accurate diagnosis can be difficult. Since cancer can spread quickly, diagnoses and prognoses must be quick and accurate for effective treatment. Diagnosis is commonly made utilizing histopathological examinations of tissue samples. The issue with these diagnoses is that they can be time-consuming and prone to errors due to the subjective nature of those examining the samples.

Cancer is characterized by the uncontrolled growth and spread of abnormal cells in the body. Given time, these cells can form tumors and quickly spread into nearby tissue and other parts of the body in a process called metastasis. Left unchecked, these tumors can interfere with the normal function of the affected organ, leading to organ failure. In the last century, we have made incredible advancements in the fight against cancer, discovering treatments that can put patients into remission. These treatments include surgery, radiation, and chemotherapy. Despite the successful use of these treatments, they can have unintended consequences. The use of chemotherapy and radiation can lead to side effects such as nausea, fatigue, and immune system suppression.

Cancer diagnoses can be an expensive and time-consuming process, due to the requirement of specially trained pathologists examining immunohistochemistry (IHC) stained biopsied tissue samples. The IHC process is very involved, requiring preserving tissue and staining via a complex protocol [18]. The pathologists will then expose the tissue samples to hormone antibodies paired with dies that bind with the predicted receptors. The dyes change the color of the cells containing the receptors thus making them identifiable [23]. This process can be subjective and can, as a result, lead to inaccurate results [8].

Fortunately, the use of modern computing power has been able to speed up the process of diagnosis for several major types of cancer, such as those that grow in the presence of estrogen (ER) and/or progesterone (PR), using machine learning models such as convolutional neural networks (CNN) like ResNet. These algorithms utilize whole slide images (WSI) of hematoxylin and eosin (H&E) stained tissue samples, which highlight cellular morphology, to train a model to classify positive and negative cancer cases. Despite the progress made using CNNs for identifying these subtypes, one issue is that there are still underrepresented subtypes of cancer, such as human epidural growth factor receptor 2 (HER2) positive and triple-negative breast cancer (TNBC), for which there are fewer available results in the literature. Several of the difficulties affecting classifying these subtypes lie in the fact that very little data is available. In addition, the tissue sample WSIs require large amounts of data storage and processing power to train the

models. Despite this, utilizing the ReceptorNet algorithm, researchers have made exceptional progress in diagnosing various cancer types [8], so we continued focusing on this methodology while exploring additional data augmentation techniques, improved feature extractors, efficiency improvement measures, and the utilization of ensemble methods. To this extent, to pay homage to the work done by [8], we have named our model ReceptorNeXt.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 presents preliminary information important to the project. Section 4 details the design and implementation of ReceptorNeXt. Section 5 presents the detailed analysis and evaluation of ReceptorNeXt through a series of experiments and performance metrics. Finally, section 6 details the conclusions and directions for future work to improve ReceptorNeXt.

2 Related Work

The area of performing image classification on cancerous tissue is being explored in many works. Techniques such as patch-wise CNN [9], using ResNet [5], ensemble methods [7], and multiple instance learning (MIL) [8] have provided fruitful results. A similarity among each of these studies is that each utilizes hematoxylin and eosin (H&E) stains, which highlight cellular morphology to determine estrogen receptor status (ERS). These H&E stained WSIs images are not annotated directly by the pathologists, since the pathologists are not trained to determine ERS from H&E stains. Instead, they are inferred by the fact that the WSI tissue samples come from tumors that were determined to have a specific receptor status, therefor the WSI must have some regions containing ER positivity if they are ER+ [8]. As opposed to using H&E staining, researchers are utilizing HER2 staining on the expensive IHC staining process and using deep learning to accurately diagnose the HER2 cancer sub-type [4]. These works have provided valuable information, upon which this work has been built.

Since the publication of their paper, significant progress has been made in the field of machine learning. Newer and more effective models such as ResNeXt have been produced, further development to residual blocks such as Squeeze and Excitation (SE) have been developed, and more efficient data processing protocols such as Automatic Mixed Precision (AMP) have emerged. To further the progress made by [8], we have implemented these new features and studied their effects.

Taking a different approach to diagnosing cancer subtypes, researchers are finding some success using logistic regression to diagnose TNBC using quantitative ultrasound image features [6] utilizing gene expression data and research using machine learn-

ing algorithms such as support vector machines (SVM), K-nearest neighbor (KNN), and Naïve Bayes (NB) to diagnose TNBC and NTNBC [12] is showing to be worthwhile.

3 Preliminaries

This section is organized as follows: subsection 3.1 discusses Multiple Instance Learning (MIL), the ML algorithm used for this work; subsection 3.2 describes the ResNet model architecture; subsection 3.3 explores the ResNeXt model architecture and how it is an improvement over ResNet; subsection 3.4 discusses attention and its benefits in ML; subsection 3.5 introduces Squeeze and Excitation blocks and their added benefit in residual networks such as ResNet; and subsection 3.6 introduces Automatic Mixed Precision (AMP) and covers its importance as datasets and models become larger and more complex. The information in this section aims to offer a comprehensive understanding of the underlying concepts used in this work.

3.1 Multiple Instance Learning (MIL)

MIL is a bag-level supervised machine learning approach that uses sets of labeled bags to learn a classifier. These bags, which contain sets of data, in our case images, are assigned a binary label based on the presence or absence of data with a particular property. A bag is labeled 'positive' if it contains at least one positive instance or 'negative' if otherwise as shown in figure 1, which shows a comparison between an ML classifier and MIL.

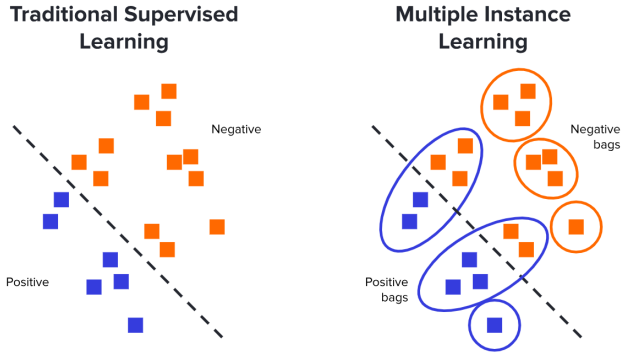


Figure 1: MIL vs traditional ML
[14]

A simple example describes MIL nicely. Imagine there are several sets of keys and there is a door that requires a key. Every set of keys that contains the key for that door's lock would be classified as positive, while those that don't, are classified as negative. The objective of the MIL model would be to determine the key that is common among all of the positive labeled sets of keys. Notice in figure 2 that the green key is the key in common with the sets that can open the door. Using MIL, the model should deduce that the green key can unlock the door, and given new sets of keys, determine whether a set can unlock the door if it has a green key or not.

(a) Negative (b) Positive (c) Positive

Figure 2: Key set example [14]

MIL is an excellent choice for cancer hormone receptor status (HRS) inference for the following reasons: The WSIs provided in the dataset are too large for standard ML algorithms to process efficiently, so it is beneficial to split them into smaller *tiles* that can quickly be loaded and processed by the model. Since it is not known which tiles contain HRS indicators, the tiles can be placed into bags that are labeled according to the label given to the WSI. The MIL model can then learn to recognize when a tile contains an HRS indicator and then identify the bag, and subsequently the WSI itself, to the proper label. Tying this back to the example with the sets of keys, each set of keys represents a single bag of tiles, where the keys themselves represent the individual tiles.

3.2 ResNet

CNNs have improved dramatically in recent years, with the development of models such as LeNet [24], AlexNet [25], and VGGNet [28]. Due to the importance of high-accuracy models for cancer diagnosis, it is important to take CNNs to the next level, It has been found that adding more layers tends to increase accuracy up to a point until over-fitting becomes an issue. Unfortunately, techniques like L2 Regularization or dropout begin to be less effective in optimization or reducing overfitting as model depth increases. As we go deeper, the efficient parameters and activations (even from the identity function) get lost in the middle because the subsequent layers fail to sustain

them. This is due to their rigorous activation through continuous updating of weights and biases [11]. Residual networks have been able to go deeper while avoiding this problem by using shortcut connections, as seen in figure 3 between every two to three layers [1]. This allows us to sustain the learning parameters of the network deeper into the neural network.

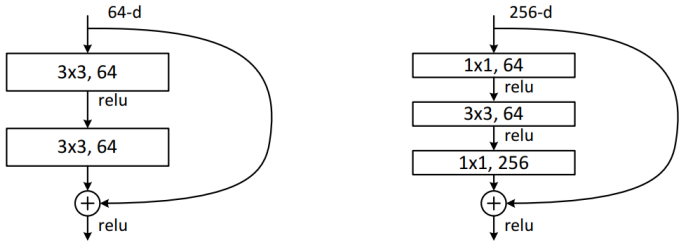


Figure 3: Left: a building block (on 56×56 feature maps) for ResNet34. Right: a “bottleneck” building block for ResNet-50/101/152 [1]

The blocks of processes between these shortcuts are called "residual blocks" and they are stacked on top of each other in the ResNET. An example of ResNet34 can be seen with its stacks of 16 residual blocks versus a 34-layer CNN in figure 4.

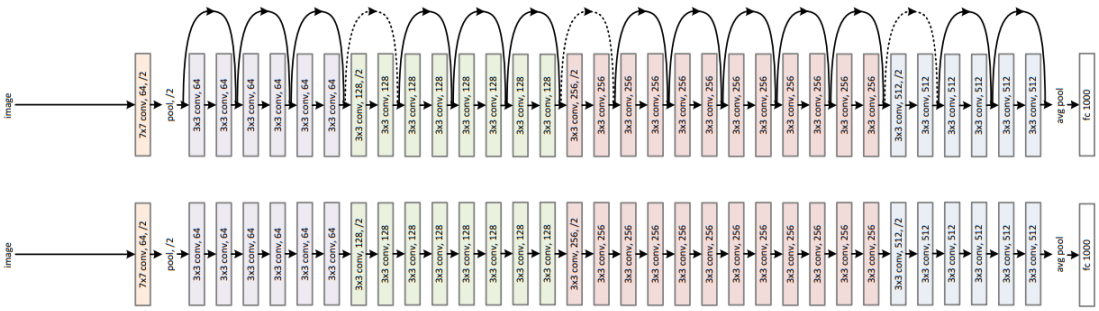


Figure 4: Top: ResNet34 architecture. Bottom: 34 layer CNN [1]

The mathematics inside these residual blocks is fairly straightforward, in that the information gets fast-forwarded deeper into the network and added back in with the processed information.

3.3 ResNext

With the success of ResNet as an advanced CNN, it is natural for modifications to be made that lead to improvements. On top of the blocks utilized in ResNet, the concept of cardinality is introduced as an additional dimension alongside depth and width. The cardinality represents a number of independent tensors, split from the input tensor, within a block, which undergoes independent transformations prior to concatenation into the block's output as shown in figure 5. The width dimension signifies the quantity of filters or channels within each parallel path. A greater width enhances the model's capability to capture more intricate patterns within each group. The depth dimension signifies the number of residual blocks in the model. For example, ResNeXt50 32x4d is a ResNeXt model with a depth of 50, cardinality of 32, and a width of 4.

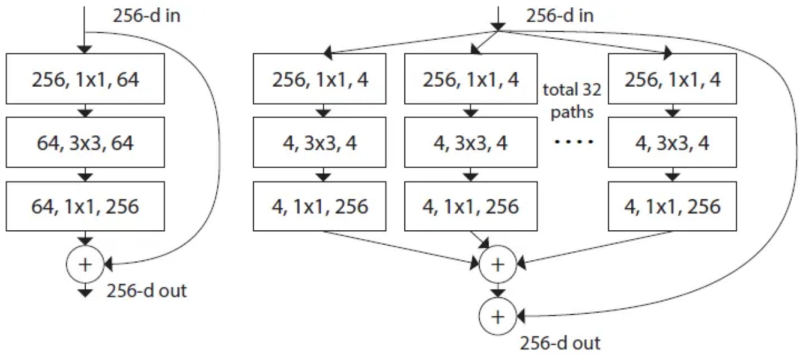


Figure 5: ResNet block vs ResNeXt block with a cardinality of 32 [2]

Results in Xie et all [2] found that increasing cardinality performed better than increasing model depth and/or width. They tested their version of ResNeXt101 against ResNet101 on the ImageNet dataset before testing against a wider variant of ResNet101 and the deeper ResNet200. As shown in figure 6, the ResNext101 model with increased cardinality achieves lower error scores than models with increased depths. Note: Top-1 error is the percentage of test samples for which the correct class is not the top predicted class. Likewise, Top-5 error is the percentage of test samples for which the correct class is not among the top 5 predicted classes.

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Figure 6: Cardinality vs depth and width in ResNet vs ResNeXt [2]

3.4 Attention

As CNN models have evolved, new problems have been found that they can be utilized for. One such area is natural language models (NLM) used for language translation. Since many languages do not utilize the same sequence of nouns and verbs, it is important to recognize what the important parts are. While NLMs are vastly different from computer-vision models, they do share the similarity in the idea that there are features of more importance that do not rely on the order in which they are fed into the model. The basic idea is that the most relevant information is concentrated instead of the entire sequence - that is, the model is able to selectively focus on the valuable parts of the input and learn the association between them [17]. When the attention model is adapted to computer-vision models, the idea is that it operates similarly to how our brain can focus on the important information in the pictures that are relayed to it by our eyes. To do this, a series of context vectors are created followed by corresponding weights, such that they add up to 1. These attention weights are calculated by normalizing the output scores of the neural network. The weights are then multiplied by the output scores to create the alignment vectors. These are then summed up to produce the context vector which is then fed into the model's decoder [17, 22].

$$c_i = \sum_{k=1}^{K} a_{ik} h_k$$

where

$$a_{ik} = \frac{\exp(e_{ik})}{\sum_{k=1}^{K} \exp(e_{ik})}$$

c_i is a context vector where:
K is the number of embeddings over which the context vector is calculated

a_{ik} is the attention weight
h_k is an embedding, which refers to the representations of input data
e_{ik} is the output score of the feed-forward neural network

Figure 7 shows, with the help of Guided-BackPropagation [19], a series of heatmaps that demonstrates the areas of an image that the model has deemed relevant to focus on with the help of an attention mechanism.

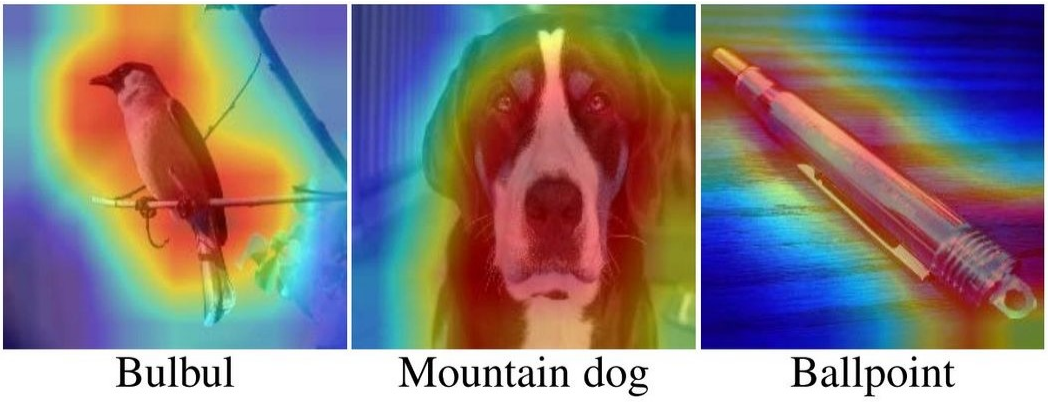


Figure 7: Example of image focus by attention mechanism [26]

Further attention practices can be explored. Utilizing spatial and channel attention, Convolutional Block Attention Module (CBAM) [26] when applied as a layer in convolutional blocks of CNNs has shown improvements on the ImageNet classification for various models.

3.5 Squeeze-and-Excitation Blocks

The data fed into a CNN model contains an enormous amount of information that can be difficult for a model to recognize what is useful. An image is composed of two basic components: spatial, the height and width of the image; and channels, often in the form of red, green, and blue filtered data. This leads to an image dimension to be in the form (h, w, {channels}), or in the case of this project, (256, 256, 3). Squeeze and Excitation (SE) blocks, as developed by [21], focus on capturing channel dependencies and recalibrate the importance of the different channels in the feature maps. This is accomplished in two steps: the squeeze and the excitation. In the squeeze step, using global average pooling channel-wise statistics are generated through the spatial dimensions. This produces an aggregated collection of local descriptors for the whole image. The

excitation step aims to capture the channel-wise dependencies aggregated during the squeezing step so that it can be used effectively. It maps the descriptors to a set of channel weights, acting as a self-attention function on the channels. This is accomplished by passing the squeeze output through a bottleneck containing two fully connected layers, a ReLu, followed by the Sigmoid function. An example is shown in figure 8. Finally, the output is scaled.

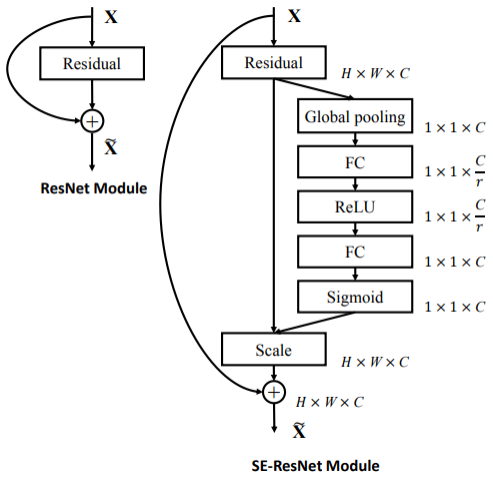


Figure 8: SE schema (right) compared to a residual module (left) [21]

3.6 Automatic Mixed Precision

As ML models grow larger and become more complex, training and evaluation runtimes will grow longer. Fortunately, there are means to reduce runtime and the overall amount of the data passed through the models. By default, Python float values are 64-bit double-precision values, or full-precision (FP). In some cases, such precision is unnecessary. Recall that numbers are represented as a series of bits. There is the signed bit, the exponent, and the significant. Figure 9 shows how the different precision numbers are represented. Again, keep in mind that Python float is double precision.

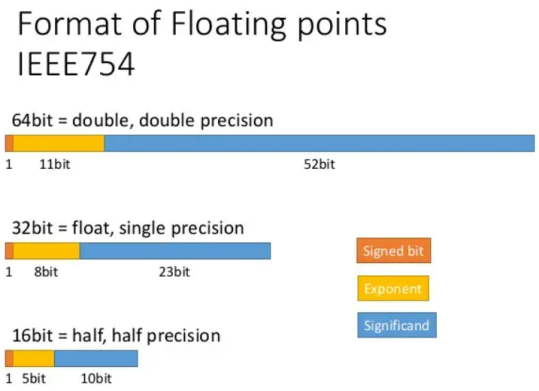


Figure 9: Double, single, and half-precision [20]

Due to the nature of binary numbers, whole numbers can be represented using relatively few digits. Decimal numbers and especially irrational numbers such as $\pi, e, \sqrt{2}$ however can require significantly more. For example, the binary representation of π is shown below in figure 10 .

Double precision	0100000000001001001000011111101101010100010001000010110100011000
Single precision	01000000010010010000111111011011
Half precision	0100001001001000

Figure 10: Precisions for binary representation of π [20]

To save memory and time during model training, data scientists experimented with training at lower precision. They realized that deep learning models have so many parameters, that the exact value of any one parameter is often not important. Despite having lower precision, most models trained in FP16 do not show any measurable performance degradation, due to their heavy over-parameterization [13]. A major drawback to training at FP16 is that it cannot represent numbers larger than 65,504 (in absolute), or smaller than 5.96e-8. To address this, there are several means to prevent issues from arising: gradient scaling allows gradients to be scaled during the backward pass to prevent underflow or overflow issue caused by working with lower precision; mixed precision, which can be used by modern GPUs, such as those running Nvidia's Turing, Volta, and Ampure chips, where calculations begin with half-precision values for rapid matrix math and the results are stored as higher precision values; and finally automatic mixed precision, which contains a predefined list of operations that are con-

sidered safe for FP16 training [13]. Overall, this technique can accelerate traditional double-precision applications by up to 25x while shrinking the memory, runtime, and power consumption required to run them [20].

4 Methods

4.1 Inspiration for the Methodology

The methods used and the results found in [8] have had a profound influence on the motivation for this research. Naik et al proposed a machine learning model that can accurately determine estrogen receptor status (ERS), a key molecular marker used for prognosis and treatment decisions in newly diagnosed breast cancer patients, directly from cellular morphology in hematoxylin and eosin (H&E)-stained whole slide images (WSI). The model, trained using a varied dataset of 3,474 patients, achieves high sensitivity and specificity and has the potential to augment clinicians' capabilities in cancer prognosis and treatment decisions by harnessing biological signals that are imperceptible to the human eye. This approach offers a dramatically less expensive, quick, and less variable alternative to the current gold standard of determining ERS through immunohistochemistry staining.

Naik et al. [8] utilize a MIL model, which they call ReceptorNet. This model is comprised of three interconnected neural networks. The first is a feature extractor, followed by an attention module, and then a decision layer. The model was trained using 50 randomly drawn tiles, which were extracted from each WSI as discussed in 3.1. Data augmentation was used to help the model learn and deal with variabilities resulting from the staining process. An overview of their model can be seen in Figure 11.

Utilizing datasets from TCGA and the Australian Breast Cancer Tissue Bank (ABCTB), they found that the results from ReceptorNet achieved an AUROC of 0.90 when testing on the training data and 0.92 when testing on the test data. They then compared their results to baseline methods by replacing their attention module with those such as Meanpool and Maxpool. They found that their attention module outperformed the baseline results of 0.827 and 0.880, respectively. It should be noted that the ABCTB dataset could not be acquired at the time of writing this report. They also explored the effects on inferring ERS when given PRS and HER2S.

While the overall goal of this project was to improve upon the results in [8] with the data available, we also wanted to provide an open-source model that could continually be improved and be capable of being used in the field. [8] does not provide their actual model, but resources that can be used to reproduce it.

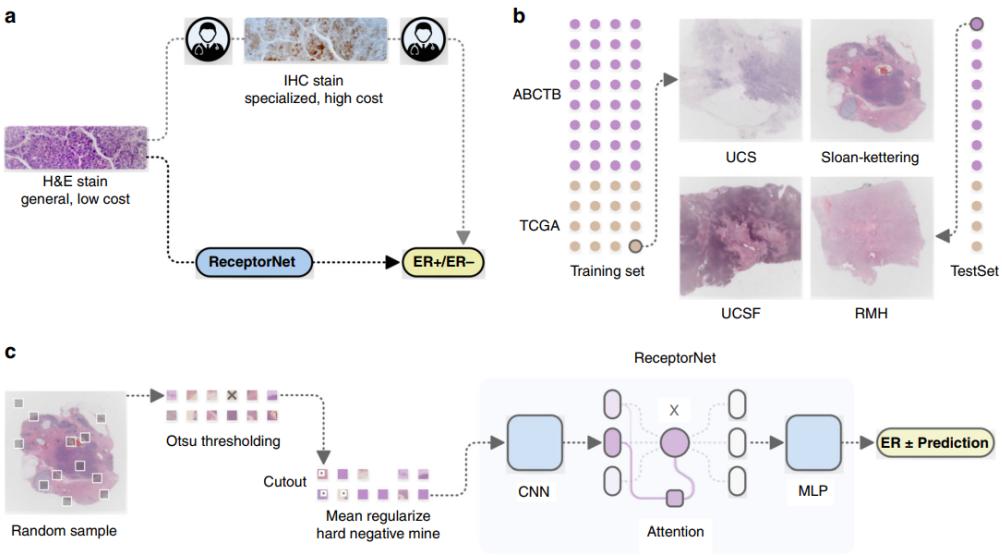


Figure 11: ReceptorNet model overview [8]

4.2 Data and Problem Description

The dataset utilized for this work is the Cancer Genome Atlas Project (TCGA). This data set is comprised of 3111 WSIs from 1098 patients. A cancer hormone receptor status is obtained from a patient's clinical information and linked to their respective WSIs. Some patients have only one WSI, while some have upwards of seven. Of these WSIs, 1,978 are marked as tissue slides and 1,133 as diagnostic slides. The latter contains significantly more data that can be used for molecular characterization of an individual's cancer [27] and occasionally contains physical annotations that could negatively impact the model. The WSIs can range in dimensions of 30k-200k pixels high and/or wide and range in size 300MB-3GB. An example WSI is shown in figure 12

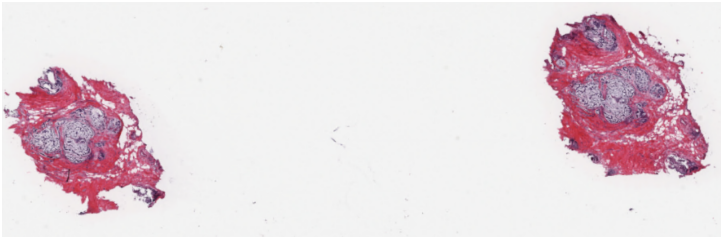


Figure 12: TCGA tissue sample WSI. Sample type: primary tumor. Annotation: ER-

Each WSI has three receptors: estrogen (ER), progesterone (PR), and human epidermal growth factor receptor 2 (HER2). Each of these has the following possible statuses: positive, negative, equivocal, and null. Correctly diagnosing these receptor statuses is an important step in providing a patient with the care they need in the fight against their cancer. After processing (see section 4.4), Separate copies of the dataset are produced for each respective receptor. For a given receptor dataset, in the cases where a WSI has a status of equivocal or null, it is removed from the dataset so that only positive and negative statuses remain. Finally, a fourth receptor category is produced called triple-negative (TNBC). This category's status is labeled positive when all three receptors (ER, PR, and HER2) are negative, otherwise a negative status is given. Table 1 shows the receptor population data after the processing is complete. It can be seen that TNBC, like HER2 has significantly fewer WSIs than ER and PR. This is due to the fact that so many HER2 and some ER and PR examples have statuses that are listed as either equivocal or null.

Label	Count	Percentage	
		Positive	Negative
ER	2555	76.7	23.3
PR	2504	66.0	34.0
HER2	1777	22.2	77.8
TNBC	1747	16.2	83.8

Table 1: Population information for receptor status

This project aims to develop an open-source data processing pipeline and ML model to acquire sets of tiles from WSIs and subsequently infer cancer hormone receptor statuses from the processed tiles. While there are multiple receptors (ER, PR, and HER2), the model is designed to be a binary classifier for positive and negative statuses of a given receptor, capable of changing its focus when desired as some WSIs may exhibit indicators from several hormone receptors. To accomplish this, the model is retrained for each respective hormone receptor.

Comparison with Naik et al. [8] The goal of this project is the same as Naik et al [8], in that we want to develop a model that is capable of accurately inferring cancer hormone receptor statuses utilizing H&E stained WSIs. Whereas they examined the effects on inferring ERS given PRS or HER2S, our goal is to accurately infer ERS, PRS, HER2S directly from the H&E stained WSIs.

With the processed data used for this model and the data used in Naik et al [8], it may be difficult to properly compare the respective AUROC scores as their TCGA dataset is

comprised of 1,014 H&E stained WSIs from 939 patients accessed in 2020 versus our dataset of 2673 H&E stained WSIs from 1,088 patients accessed in 2023. Currently, the total TCGA dataset consists of 3,111 WSIs from 1098 patients. There is a dramatic difference between the size of the two datasets, so it would seem that they were either very selective of the data they used or the dataset has grown in this time.

4.3 Experimental System

4.3.1 Data processing machine(see Section 4.4)

Dell PowerEdge R6425, 2x AMD EPYC 7H12 64-Core 2.6Ghz CPU, 512GB RAM, Fedora Linux 38 (Server Edition)

This machine is generally reserved for student virtual machines, so we were given an allotment of 200 processors out of the total 256 available.

4.3.2 Model training machine (see Section 4.5)

Custom build with: AsRock X399 Taichi sTR4 ATX Motherboard, AMD Ryzen Threadripper 2950X, 16-core CPU 64GB DDR4 2400 RAM, RTX 2080Ti 11GB, CentOS Linux 7 (Core).

4.4 Data Processing

The WSIs were processed into 256x256 pixel tiles utilizing the OpenSlide Python library. The infrastructure involved in photomicrography that OpenSlide handles is that of a pyramid structure. At the base level is the original image at full resolution and magnification. Subsequent layers contain progressively zoomed-out images, down-sampled to lower resolutions [10].

The dataset contains a mix of WSIs that were photographed at different magnifications. To account for this, all WSIs imaged at 40x magnification were down-sampled to the next lower level as demonstrated in figure 13 to ensure that all images are an equivalent magnification. An example of the effect on a tile at the different zoom levels can be seen in figure 14.

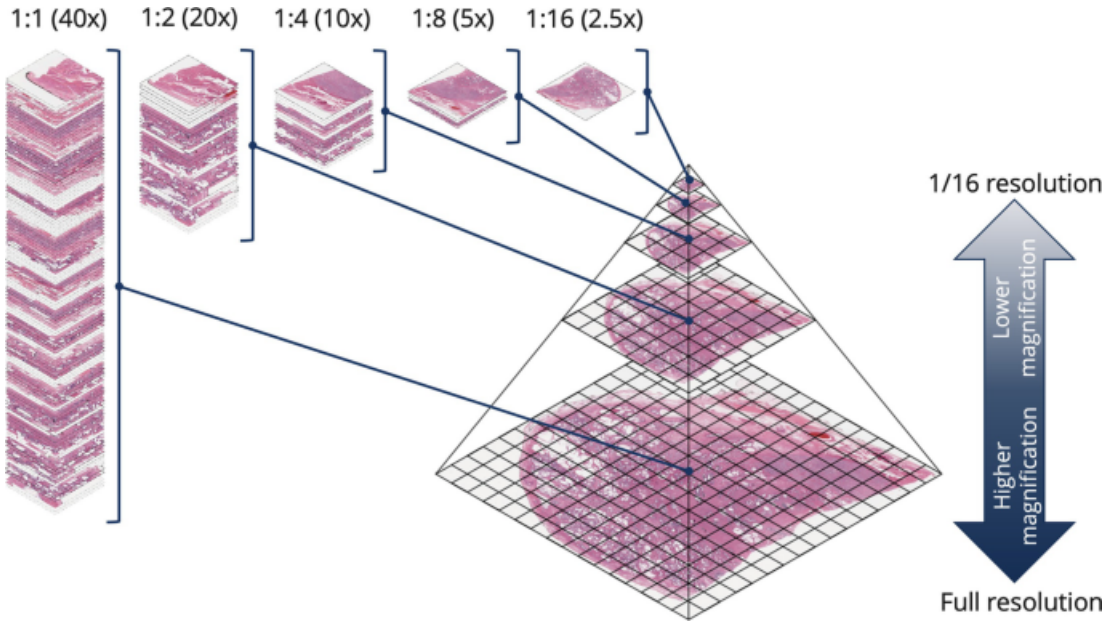
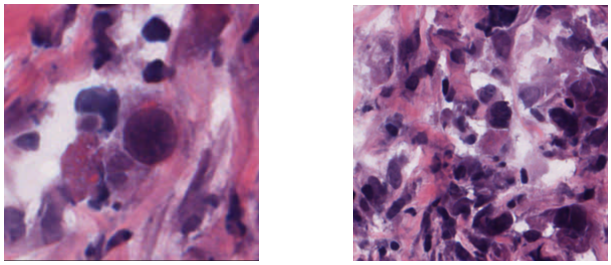


Figure 13: WSI pyramid levels



(a) 40x zoom tile (b) 20x zoom tile

Figure 14: Objective magnification

Initial work on the preprocessing algorithm is credited to Stefano Mozart and Vinicius Navega Stelet with the Brazilian Institute of Cancer. Their work was combined and refined to build and process the dataset. Due to the large size of the WSIs, processing a single image into tiles and saving them can take upwards of 20 minutes. With such a large amount of data, this can lead to several days to weeks to fully process the dataset. To reduce the preprocessing time, several actions were taken:

- Multiprocessing was utilized to split each image into smaller sections which could then be processed in parallel utilizing the available CPU affinity of the processing machine as described in Section 4.3.1.
- Due to the nature of the tissue slide images, there is significant white space that yields no useful data. Recall that an image is made of three color channels, [R, B, G] and each has a value that ranges from 0 to 255. A completely white image is [255, 255, 255]. so a threshold was set such that tiles with an average pixel value across all three channels above 180 would be rejected.
- There are instances where some tissue material is not well focused on and appears blurry. To account for blurry tiles, each tile was converted to grayscale before taking the LaPlacian to find its gradients. The resulting gradient was compared to a threshold value of 100 to determine whether it was blurry. Those that fell under the threshold were deemed blurry and rejected.
- There are locations on the WSI where shadows are captured that fall under the white space average pixel data threshold. To account for these tiles which contain nothing but flat shadow tiles, the standard deviation of the average pixel value for each of the red, green, and blue channels was calculated and compared against a threshold of 10. The tiles with values under the threshold were rejected.
- Finally, since very few, if any of the WSIs' dimensions are multiples of 256, some tiles, which have non-square 256-pixel dimensions, are the remaining pieces found on the edges of the WSI segments described above from multiprocessing. These tiles are rejected to ensure that the data fed into the model are all of consistent dimensions.

Once each WSI is processed into tiles, the file path for each respective tile is registered and stored in a CSV file for reference by the model, and the original WSI file is removed to save space. Finally, due to the tile rejection threshold and the nature of the WSIs, there may be cases where very few tiles were acquired. In this case, since the model may be trained with 30-50 tiles per bag (see section 4.6), WSIs that yielded fewer

than 100 usable processed tiles were rejected and removed. The processed data would then be transferred to the training machine. The data was processed in chunks to allow for the training model to be refined while more data was being processed.

Unfortunately, as a result of the filtering procedure, 395 WSIs were rejected. Figure 15 shows the distribution of tiles for each WSI after processing. If this model was put into production, if any tissue samples that were to be tested were rejected, they would need to be recollected and re-stained.

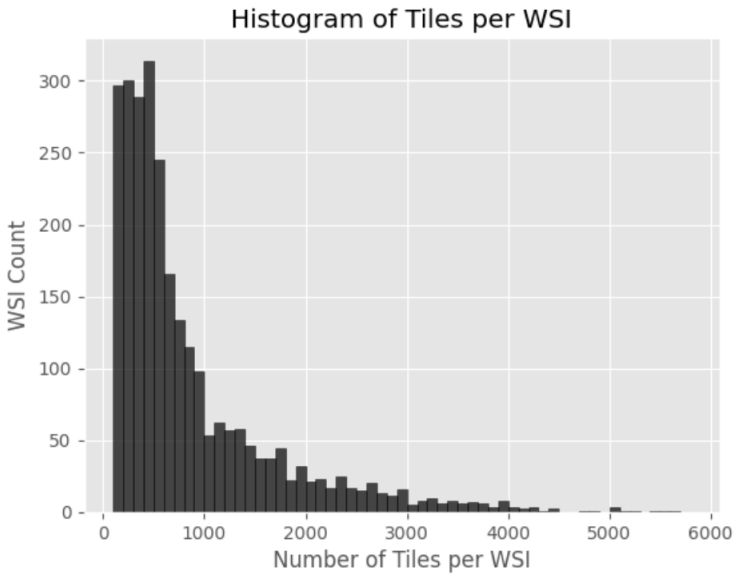


Figure 15: Histogram of WSI tile count in bins of 100

Comparison with Naik et al. [8] Their paper describes the data preparation process as using Otsu thresholding to remove background regions. We alternatively removed background regions by examining the color channel brightness of tiles to determine what was tissue and what was not. We similarly extracted 256x256 pixel tiles at 20× magnification without any overlap. We additionally examined the sharpness of the tile image, rejecting when the Laplacian of the greyscale converted tile is below a desired threshold.

4.5 Model

The model architecture was initially built in the same manner as that of [8]. A feature extractor, an attention module, and a decision module.

The feature extractor is a ResNeXt-50 w/ SE with two fully connected linear layers with a dropout of 0.5. The ResNext-50 is initialized from ImageNet V2 pretrained-weights. Each of the fully connected linear layers is randomly initialized using He initialization.

Just as in the model by Naik et al [8], during one iteration of training, a bag consisting of N tiles is fed into the feature extractor, similarly to how words in a sentence are fed into an LLM. The feature extractor then outputs an $N \times 512$ feature matrix which is fed into the attention module for aggregation.

The attention module features a linear layer that reduces each feature vector to 128 dimensions before applying a hyperbolic tangent on the input. This scales the features to include values between -1 and 1 and facilitates the learning of similarities and differences between tiles. The output is then fed into another linear layer and a Softmax function which computes an attention weight between 0 and 1 for each tile. With this, an N-dimensional vector of attention weights is produced. The inner product of this vector of attention weights and the $N \times 512$ output of the feature extractor is taken to produce a 512-dimensional feature vector. The attention mechanism ideally gives greater weight to tiles that would be assigned a positive label (key instances). It does not provide a prediction probability, but it could be considered as a proxy to that [22].

The feature vector is then fed to the decision layer which contains a 512-dimensional linear layer followed by a Sigmoid function that outputs a probability between 0 and 1.

Experiment Design Details A series of feature extractors were explored. These feature extractors include:

- ResNet50 model sourced from PyTorch TorchVision models [1]
- ResNeXt50_32x4d model sourced from PyTorch TorchVision models [1]
- ResNeXt50_32x4d with SE model sourced from Hugging Face models [2]
- ResNet101 model sourced from PyTorch TorchVision models [1]
- ResNeXt101_32x4d model sourced from Torch Hub models [3]

All of the ResNet and ResNeXt feature extractors tested required no alteration to the rest of the model.

[1] https://pytorch.org/vision/0.9/models.html
[2] https://huggingface.co/docs/timm/models/seresnext
[3] https://pytorch.org/hub/

Additional experiments were carried out to judge the effects of using ensemble methods with various bag sizes. The ensembles were created by rerunning evaluations on the test dataset multiple times and aggregating their inferences. These ensemble tests involved gradually increasing the bag sizes from 50 tiles to as many as the GPU listed in Section 4.3 could handle for a given model. Their respective inference times and AUROC score improvements were then recorded (see tables 5)

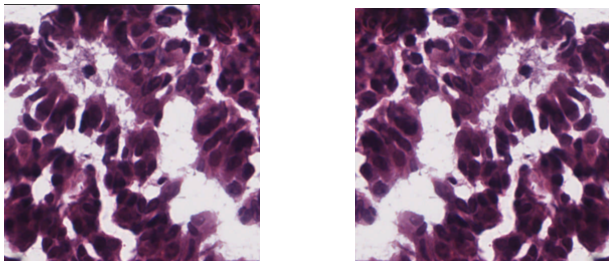
Comparison with Naik et al. [8] Their paper describes using a standard ResNet50 for feature extraction. It is not directly stated what the number of tiles were included in each bag for evaluation, but judging from the fact that they used Nvidia P100 GPUs, which have 16GB of memory for model training compared to ours as described in section 4.3.2, it was assumed that they were able to use larger bag sizes for evaluation. In their paper, they also describe 'sampling multiple bags from a WSI and aggregate their probabilities to improve inference accuracy'. We did similarly in running multiple validations/tests while resampling the tiles each time and aggregating the results to make an inference.

4.6 Training Protocol

Due to a sizeable imbalance of the data, balanced under-sampling was performed on the training dataset at the WSI level to ensure that the model had equal access to positive and negative cases. The model is trained by feeding bags of 50 randomly drawn tiles from each WSI. 50 tile bags are used to prevent overfitting as implemented by Naik et al [8].

Data Augmentation Data augmentation was used to effectively expand the number of tiles available for each training bag corresponding to a WSI, deal with differences between staining methods between WSIs, and help learn invariances. While randomly choosing tiles for a bag, each time a tile was chosen, it would undergo each of the data augmentations listed below. The specific data augmentation methods include the following:

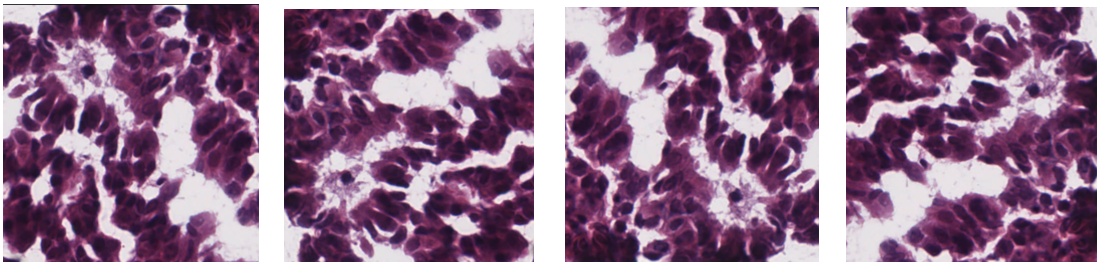
1. Randomly flipping a tile horizontally with a probability of 0.5.



(a) Raw image (b) Horizontal flip

Figure 16: Random flip

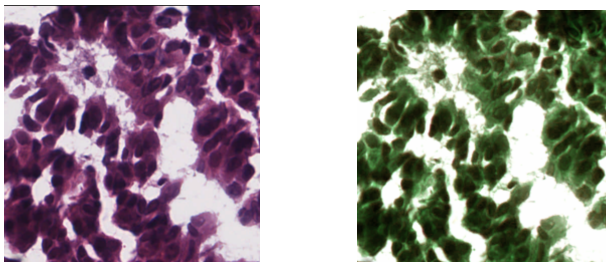
2. Randomly rotating a tile by 0, 90, 180, or 270 degrees with equal probability.



(a) Raw image (b) 90 degree (c) 180 degree (d) 270 degree

Figure 17: Random rotation

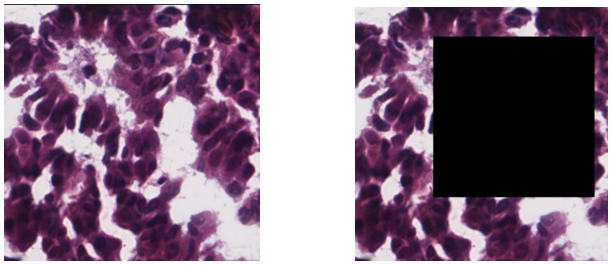
3. Performing color jittering.



(a) Raw image (b) Color jitter

Figure 18: Color jitter

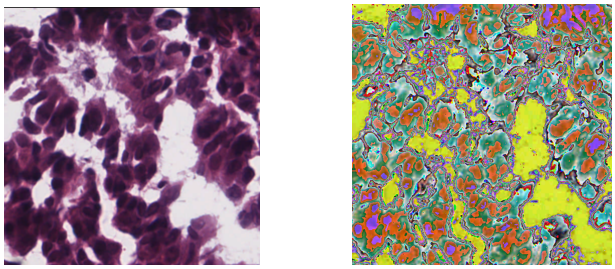
4. Random cutout regularization (random erasing) with a length of 100 pixels and 1:1 ratio



(a) Raw image (b) Cutout

Figure 19: Cutout regularization

5. Normalization



(a) Raw image (b) Normalized

Figure 20: Normalization

6. Alternatively, replaced tile with tile comprised of the dataset's average pixel value with a probability of 0.75 and the above applied

Figure 21: Dataset average pixel value

Model Operation To achieve the quickest loading times, the data loader was initialized utilizing the optimal number of workers. The optimal number was determined us-

ing the algorithm in Anwar [15]. Due to the positive/negative classification, the model utilized the binary cross-entropy loss function and Adam optimizer with a learning rate of 1×10^{-5} and a weight decay of 5×10^{-5}. To improve efficiencies with respect to both training time and memory usage, AMP was employed utilizing Pytorch's Grad Scaler described in [16]. For AMP to function correctly, the loss function must specifically utilize binary cross-entropy with logit loss. This loss function operates similarly to binary cross-entropy loss, except that it has a built-in Sigmoid function, therefore it requires that the model's classifying layer skips the Sigmoid function when calculating the losses.

For the purpose of building the model, training was conducted on ER labels. Later, the model would be retrained for the other hormone receptors. Since there is a large imbalance of positive and negative classifications for each of the hormone receptors, undersampling was used to provide balance. A set of the overall training data was sampled to ensure an even balance of positive and negative values. Before each epoch, the data of the majority class to be fed into the model was resampled to ensure that new information was being introduced.

The number of epochs was configured to be 300, however, an early-stopping mechanism was put in place. Over the course of the experimentation process, the early-stopping mechanism required changing. The initial early-stopping mechanism was initialized by a patience value and a loss delta value as shown in algorithm 0. Throughout the training process, any time the validation loss exceeded the preset limit shown below, a counter was increased until it reached the patience limit as shown in 0. In practice, the patience was set to 10 and the delta value to 0.6.

$$loss_{val} \geq loss_{min} + \Delta$$

Where $loss_{val}$ is the validation loss in the current epoch
$loss_{min}$ is the lowest loss of all the epochs up to the current
Δ is a set differential between $loss_{val}$ and $loss_{min}$ that triggers the event.

After experimenting with the different feature extractors, the better performance led to a new issue that was leading to longer training times. As the feature extractors led to better learning models, the losses due to over-fitting decreased in magnitude as time went on causing the original early spring conditions to never trigger. The model's performance was not improving but the losses were not extreme. To address this, the early-stopping condition was changed to take a ten-epoch rolling average for the training and validation losses and then take the difference between the two. If the difference between the two was greater than a defined value, then early stopping would trigger. Since the losses can be erratic at the beginning of the model, the condition would

not trigger within the first 45 epochs. It was designed such that these values could be changed depending on the user's desire.

Algorithm 1 Early Stop Class:

Input: Two numbers P and D
Let t_{losses} be an empty list ▷ Hold training loss values
Let v_{losses} be an empty list ▷ Hold validation loss values
$p \leftarrow P$ ▷ p and P is the user initialized patience
$\Delta \leftarrow D$ ▷ Δ and D is the user initialized allowable loss difference
$min_{loss} \leftarrow \infty$ ▷ Initialize a minimum value for the validation loss
$c \leftarrow 0$ ▷ Initialize a counter

Algorithm 2 Early Stop definition in class

Input: Two numbers V_{loss} and t_{loss}
Output: boolean
$t_{losses} \leftarrow t_{losses} + t_{loss}$
$v_{losses} \leftarrow v_{losses} + v_{loss}$
if $t_{losses} > 10$ **then**
    Remove $t_{losses}[0]$
    Remove $v_{losses}[0]$
end if
if $v_{loss} < min_{loss}$ **then** ▷ New low v_{loss}, reset counter
    $min_{loss} \leftarrow v_{loss}$
    $c \leftarrow 0$
end if
if $v_{loss} > min_{loss} + \Delta$ **then** ▷ v_{loss} too high, increment count
    $c \leftarrow c + 1$
    if $c \geq p$ **then** ▷ Out of patience, stop training
        Return True
    end if
end if
if avg(v_{loss})−avg(t_{loss}) > Δ and epoch> 50 **then** ▷ v_{loss} diverging too far, stop training
    Return True
end if

To further aid in increasing accuracy and more importantly AUROC, hard-negative

mining was employed. Depending on the balance of the data, in instances where the model falsely predicts a classification as the larger population for those that are in the smaller population, the loss is counted twice to reinforce the importance of correctly labeling lower population cases. For example, examining the ER data, there are significantly more ER- instances than ER+. When the model incorrectly predicts a negative instance to be positive, the loss is counted twice.

Utility Metric Performance was measured using the area under the receiver operating characteristic curve (AUROC). An ROC curve is a graph showing the performance of the classification model and is comprised of two parameters:

True positive rate, where TP is the number of true positives and FN is the number of false negatives:

$$TPR = \frac{TP}{TP + FN}$$

False positive rate, where FP is the number of false positives and TN is the number of true negatives:

$$FPR = \frac{FP}{FP + TN}$$

The ROC plots TPR vs FPR over a series of thresholds tested against the inference probabilities. An AUROC of 0.5 indicates that the model is as effective as making random guesses.

4.7 Validation and Testing the Model

Prior to training, the dataset was split into training and test datasets for each respective HRS utilizing a 20% stratified split to ensure equal representation of the data. To obtain a general performance metric for the model over the dataset, k-fold cross-validation was used on the training dataset. At first, it was decided to use 10 folds as opposed to the 5 folds used by Naik et al [8] to ensure an increased training data size at the expense of the overall cross-validation runtime for the feature extractor experiments. To evaluate the model on the HRSs, we utilized the model with the best-performing feature extractor. It was decided to revert to 5-fold cross-validation so the results could be better compared with those of Naik et al [8] and ensure that we would not run out of time. After cross-validation, the model was retrained on the training dataset and then evaluated on the test dataset. The evaluation data, when fed into the model, undergoes no data augmentation so that its classification can be made using clean and unaltered information.

While the training bag sizes contained 50 tiles, the validation and evaluation bags contained 500 tiles. The increased bag size for validation was chosen to allow more data to be utilized and allow the maximum amount of data that the GPU, listed in 4.3, could contain in its memory before crashing.

Comparison with Naik et al. [8] As previously stated, in their model, they implement 5-fold cross-validation. Their paper does not specifically state the number of tiles per bag used during validation or evaluation. There is a statement where they describe training on individual tiles and then evaluating on an aggregation of 50 tiles, but it is not stated whether this evaluation procedure is the standard in which they operated.

5 Results

5.1 Feature Extractor Cross-Validation

Table 2 shows the average, minimum, and maximum fold AUROC scores, as well as runtimes for training and validating the model per epoch. For each epoch, 1,060 WSI were sampled for training and 251 WSIs were used for validation. The total model training runtime can be extrapolated over the number of epochs needed for training and validation. The models in these experiments utilize AMP to improve efficiency as shown in table 5.2.

Model	Fold AUROC			Runtime (min)	
	Avg	Min	Max	Train	Val
ResNet50	0.774	0.643	0.853	2:17	1:27
ResNeXt50	0.798	0.760	0.839	2:35	1:34
ResNeXt50 w/ SE	<u>0.818</u>	0.786	0.866	3:04	1:46
ResNet101	0.806	0.775	0.850	3:15	2:03
ResNeXt101 w/ SE	0.746	0.716	0.782	4:27	2:28

Table 2: Cross-validation results

Table 2 shows that the ResNext50 with squeeze and excitation blocks feature extractor performed the best of those that were tested. It was surprising that despite the fact that the ResNet101 feature extractor performed better than that of ResNeXt50, ResNext101 w/ SE performed so poorly, as seen in the appendix. With more time available, hyper-parameter tuning could have allowed for better results for the individual models.

5.2 Automatic Mixed Precision (AMP)

Utilizing the best-performing model in Table 2, to show the difference between models utilizing FP and AMP, timed tests were carried out to track the run times and GPU memory utilization across several bag sizes. The training was kept to 50 tiles per bag, while validation bag sizes ranged from 50 to 500 tiles. The training and validation data for this test was acquired from the fold 1 stratified split from cross-validation of the ER dataset.

Precision	Runtime (min) for bag sizes							Max Memory Allocated (MiB)
	Train	Validation						
	50	50	100	200	300	400	500	
FP32	6:26	0:27	0:55	2:26	4:10	5:44	10:40	10477
AMP	3:04	0:15	0:30	1:28	2:26	2:38	3:22	9114

Table 3: Effectiveness of AMP on ResNeXt50 w/ SE model

It can easily be seen just how effective utilizing AMP is on the model Examining Table 3, the model utilizing AMP can train and validate significantly faster by a factor of 2-3 depending on the number of tiles used and utilize nearly 1 GB less memory in the case of this model, which might allow for the opportunity to use more complex models exhibiting deeper, wider, and/or higher cardinality features.

5.3 HR Inference

Utilizing the best model architecture from Table 2, the model was retrained for each of the aforementioned hormone receptors. Validation and testing utilized 500 tile bags to ensure a faster runtime and to set baselines for the ensemble tests. Table 4 shows the average 5-fold cross-validation and test data evaluation scores. Accuracy, AUC, sensitivity, and specificity are reported.

Subtype	Cross-Val	Test
	Avg. AUC	AUC
ER	0.789	0.803
PR	0.734	0.760
HER2	0.650	0.612
TNBC	0.731	0.582

Table 4: Cancer HRS Inference Results

Examining the results acquired from the tests on the various HRS, it is immediately apparent that the model was unable to achieve the results found in Naik et al [8] for ERS. That being said, we will examine the results for each HR individually as shown in table 4.

5.3.1 ER

On cross-validation using only TCGA data, Naik et al [8] report an AUC of 0.861, while we were only able to achieve 0.789 in cross-validation. Despite our best attempt to reproduce their model as closely as possible, there may certainly be differences in the model that led to a poorer performance. The size of the respective datasets varied dramatically, in that we used over 1600 more WSIs. One might assume that this would provide an advantage to our model, but it did not work out that way.

5.3.2 PR

While not as effective as with ERS in cross-validation, the model performed comparably well on a test dataset. With an AUC of 0.76 on the test data, it can be said that the model performs fairly well.

5.3.3 HER2

The model was poor in directly inferring HER2 status. Whether this is due to the effectiveness of the model, the amount and/or quality of the data available, or a combination of the two, at this time, it is not recommended to use this model for HER2 inference as it is only just marginally better than guessing the correct status based on H&E stained WSIs.

5.3.4 TNBC

The model proved surprisingly effective at inferring TNBC statuses in cross-validation considering how poorly the HER2-trained model performed. Evaluating the test dataset proved to be disappointing. Perhaps the data in the test set happened to be more difficult cases.

5.3.5 Discussion

While the model did not perform as well as that of the model in Naik et al [8], it is immediately apparent how important it is to have as much data as possible as the cross-validation results of the 10-fold model, used for testing the feature extractors on ER

data, and 5-fold model, used for comparing the subtypes when testing on the ER data, differed by nearly three points. Additionally, since some WSIs provided fewer tiles than others, there may not have been enough data containing the HRS indicators. Overall, the model demonstrated the ability to discriminate the statuses well in three of the four hormone receptors in cross-validation and shows promise that improvements can be made with more time.

Comparison with Naik et al. [8] It is important to reiterate the fact that we tested the model specifically to make inferences on the different hormone receptors, whereas Naik et al examined the effects of an HR such as PR or HER2 on ERS. For example, they found that their model performed better in inferring ERS on HER2- than HER2+.

5.4 Ensemble Inference

To observe the effects of utilizing fewer tiles per bag with multiple runs as an ensemble (aggregating multiple bag inferences), tests were carried out utilizing the model with the best-performing feature extractor as shown in Table 2 with the 511 WSI test dataset separated before training (20% of the total population). The tests were run using bags of 50, 100, 250, and 500 tiles. For each bag size, the number of evaluations, and subsequently ensemble members, were dependent on the number of runs needed to accrue the tiles necessary to reach the size of the largest bag. That is, the 50-tile bag inference was run ten times, the 100-tile bag inference was run five times, the 250-tile bag inference was run twice, and the 500-tile bag inference was run once. Since the 500-tile bag was run once, it was considered to be the baseline for comparison to the other bag-size ensembles.

The ensemble method was performed as follows: After each inference round, the results were saved to a new column in a data frame. After the final inference round, the results of the rounds for each of the respective WSI are then examined. A MAX decider is used with the assumption that there may be situations where a bag of tiles may not contain any positive hormone receptor indicators for inference, so if another bag does contain such tiles, then the inference on the WSI must be positive. The maximum value in each row corresponding to a WSI is then selected and returned as the official inference probability. To account for irregularities in the inference process and tile selection, each ensemble was run five times and the averages are reported. Table 5 shows the improvements of the ensembles over the 500 tile bag's AUC. The ensemble experiment was conducted for each of the HRs.

Tiles	AUC Imp	Time (min)
50	-1.65	5:00
100	-1.05	4:46
250	-0.20	4:10
500	—	3:31

(a) ER

Tiles	AUC Imp	Time (min)
50	-2.40	5:08
100	0.00	4:46
250	1.00	4:13
500	—	3:39

(b) PR

Tiles	AUC Imp	Time (min)
50	3.20	3:59
100	2.70	3:28
250	1.30	3:06
500	—	2:36

(c) HER2

Tiles	AUC Imp	Time (min)
50	5.10	5:46
100	4.70	5:04
250	4.40	4:33
500	—	3:58

(d) TNBC

Table 5: Ensemble Results on HRs

Examining the results in Tables 5, it can be seen that utilizing this ensemble method shows a lot of promise for increasing the model's performance for cases where positive instances are in the minority. It can also be seen that it is more efficient in runtime to use larger bag sizes, so when training a model, it may be more desirable to utilize larger bag sizes and then test with the ensembles. Based on these results, the use of these ensembles would be valuable for systems using GPUs with limited memory that cannot load the entire dataset. Also, situations where the dataset is heavily imbalanced with positive cases being the majority do not perform well as the number of false positives increases.

6 Conclusions

The need for more efficient and accurate ways of diagnosing cancer is ever-growing for doctors to be able to provide quick and effective treatment. The utilization of machine learning models like CNNs trained on H&E WSIs to provide a viable alternative to the expensive and subjective IHC diagnosis process is becoming a real possibility.

This report has detailed the methods and process of building such a model and data processing pipeline capable of utilizing H&E stained tissue samples collected by TCGA. We have produced a data processing pipeline that takes the manifest data acquired from TCGA, builds a dataset containing all necessary classification data, downloads the WSIs, processes the images into tiles, filters out unwanted tiles, and updates the dataset

with their respective file locations.

Continuing the work described in Naik et al [8], we have successfully built a model, which we have named ReceptorNeXt, that implements AMP to reduce training time and memory requirements and utilizes the newer and more effective feature extractor. We have shown that the model is capable of correctly inferring HRS to a degree that shows promise in further exploration. After experimenting with several feature extractors, the best was found to be ResNeXt50 with squeeze and excitation blocks. It achieved the best AUROC score and does not require an exceptionally long time to train and evaluate on ER data. Experimenting with AMP, we have shown it to be significantly more efficient, running over three times faster when evaluating the model on a bag size of 500 tiles. Utilizing ensemble methods to aggregate more tiles than the available GPU4.3.2 could handle at once proved to be effective, raising the AUC score by upwards of six points. To more effectively evaluate the model, it would be extremely helpful to acquire more data and utilize a more powerful, and perhaps multiple, GPU machine.

There are several interesting directions for future work that we believe could improve efficiency and/or accuracy. These include:

1. Explore Nvidia DALI to improve data load times
2. Explore Dual-Path networks [3] utilizing ResNe(X)t and DenseNet feature extractors to extract more useful information
3. Explore efficient pruning methods to improve model size and accuracy
4. Obtain more data for training the model. Data sources in mind include:
   (a) The ABCTB dataset used by [8]
   (b) A new dataset being built by NIC Brazil which contains an expanded set of classifications for rare cancer subtypes
5. Utilize a multiple GPU cluster to parallelize the training and testing process
6. Explore the use of multi-label classification. Though some of the ER and PR data would have to be ignored due to null and equivocal entries, the need for four separate models could be replaced with a single model
7. Configure model to run in Docker container for easy deployment

7 Acknowledgement

I would like to thank my committee chair Anderson Nascimento and members Juhua Hu and Martine De Cock for their invaluable insight and advice throughout this project. Furthermore, with several members of my family having been diagnosed with and beating breast cancer, I would like to thank Anderson for introducing me to this project and providing the opportunity to work on such an important and impactful endeavor focusing on cancer inference. I would also like to thank Stephen Rondeau from the School of Engineering and Technology for his support in providing the computational resources and his assistance with technical issues that were so crucial to the success of this project. Finally, I would like to thank my beautiful wife for her support throughout this journey, especially with the birth of our first child two months ago.

References

[1] Kaiming He et al. *Deep residual learning for image recognition*. Dec. 2015. URL: https://arxiv.org/abs/1512.03385.

[2] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *CoRR* abs/1611.05431 (2016). arXiv: 1611.05431. URL: http://arxiv.org/abs/1611.05431.

[3] YunPeng Chen et al. *[1707.01629v2] Dual Path Networks*. July 2017. URL: https://arxiv.org/abs/1707.01629v2 (visited on 01/01/2024).

[4] Michel E. Vandenberghe et al. "Relevance of deep learning to facilitate the diagnosis of HER2 status in breast cancer". In: *Scientific Reports* 7.1 (2017). DOI: 10.1038/srep45938.

[5] Mehdi Habibzadeh Motlagh et al. "Breast cancer histopathological image classification: A deep learning approach". In: (2018). DOI: 10.1101/242818.

[6] Tong Wu et al. "Machine learning for diagnostic ultrasound of triple-negative breast cancer". In: *Breast Cancer Research and Treatment* 173.2 (2018), pp. 365–373. DOI: 10.1007/s10549-018-4984-7.

[7] Zabit Hameed et al. "Breast cancer histopathology image classification using an ensemble of Deep Learning Models". In: *Sensors* 20.16 (2020), p. 4373. DOI: 10.3390/s20164373.

[8] Nikhil Naik et al. "Deep learning-enabled breast cancer hormonal receptor status determination from base-level H&;E stains". In: *Nature Communications* 11.1 (2020). DOI: 10.1038/s41467-020-19334-3.

[9] Rui Yan et al. "Breast cancer histopathological image classification using a hybrid deep neural network". In: *Methods* 173 (2020), pp. 52–60. DOI: 10.1016/j.ymeth.2019.06.014.

[10] David S. McClintock, Jacob T. Abel, and Toby C. Cornish. "Whole Slide Imaging Hardware, Software, and Infrastructure". In: (2021). URL: https://link.springer.com/chapter/10.1007/978-3-030-83332-9_2.

[11] Vanshika Sharma. *Resnets: Why do they perform better than classic convnets? (conceptual analysis)*. Oct. 2021. URL: https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53.

[12] Jiande Wu and Chindo Hicks. “Breast cancer type classification using machine learning”. In: *Journal of Personalized Medicine* 11.2 (2021), p. 61. DOI: 10.3390/jpm11020061.

[13] Ben Snyder J. *Solving the Limits of Mixed Precision Training | by Ben Snyder | Medium*. Feb. 2023. URL: https://medium.com/@jbensnyder/solving-the-limits-of-mixed-precision-training-231019128b4b (visited on 01/15/2024).

[14] *An Introduction to Multiple Instance Learning - NILG.AI*. URL: https://nilg.ai/202105/an-introduction-to-multiple-instance-learning/ (visited on 11/09/2023).

[15] Talha Anwar. *PyTorch num_workers, speedy training*. URL: https://chtalhaanwar.medium.com/pytorch-num-workers-a-tip-for-speedy-training-ed127d825db7 (visited on 2023).

[16] *Automatic Mixed Precision for Deep Learning | NVIDIA Developer*. URL: https://developer.nvidia.com/automatic-mixed-precision (visited on 11/16/2023).

[17] Nagesh Singh Chauhan. *Attention mechanism in Deep Learning, Explained - KDnuggets*. URL: https://www.kdnuggets.com/2021/01/attention-mechanism-deep-learning-explained.html (visited on 11/13/2023).

[18] *Chromogenic IHC Staining Protocol of Paraffin-embedded Tissue Sections: R&D Systems*. URL: https://www.rndsystems.com/resources/protocols/protocol-preparation-and-chromogenic-ihc-staining-paraffin-embedded-tissue (visited on 10/26/2023).

[19] *Deep Learning: Guided BackPropagation*. URL: https://leslietj.github.io/2020/07/22/Deep-Learning-Guided-BackPropagation/ (visited on 01/14/2024).

[20] *Difference Between Single-, Double-, Multi-, Mixed-Precision | NVIDIA Blog*. URL: https://blogs.nvidia.com/blog/whats-the-difference-between-single-double-multi-and-mixed-precision-computing/ (visited on 11/27/2023).

[21] Jie Hu et al. *Squeeze-and-Excitation Networks | IEEE Conference Publication | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/8578843 (visited on 11/06/2023).

[22] Maximilian Ilse, Jakub Tomczak, and Max Welling. *[1802.04712] Attention-based Deep Multiple Instance Learning*. URL: https://arxiv.org/abs/1802.04712 (visited on 11/13/2023).

[23] *Immunohistochemistry: IHC Test for Cancer*. URL: https://www.cancercenter.com/diagnosing-cancer/diagnostic-procedures/immunohistochemistry (visited on 10/26/2023).

[24] Ji-Chao Jiao. *[2301.04275] LENet: Lightweight And Efficient LiDAR Semantic Segmentation Using Multi-Scale Convolution Attention*. URL: https://arxiv.org/abs/2301.04275 (visited on 01/15/2024).

[25] Alex Krizhevsky. *ImageNet classification with deep convolutional neural networks | Communications of the ACM*. URL: https://dl.acm.org/doi/10.1145/3065386 (visited on 01/15/2024).

[26] Diganta Misra. *Convolution Block Attention Module (CBAM) | Paperspace Blog*. URL: https://blog.paperspace.com/attention-mechanisms-in-computer-vision-cbam/ (visited on 11/26/2023).

[27] *Projects - GDC Docs*. URL: https://docs.gdc.cancer.gov/Data_Portal/Users_Guide/Projects/ (visited on 12/16/2023).

[28] Karen Simonyan. *[1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition*. URL: https://arxiv.org/abs/1409.1556 (visited on 01/15/2024).

A Project Rsources

A.1 Code Availability

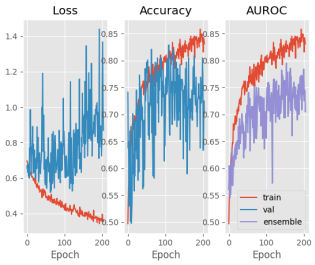
The data preparation and model code can be acquired at `https://github.com/deanak1987/TCSS702_Capstone`

A.2 Data Availaility

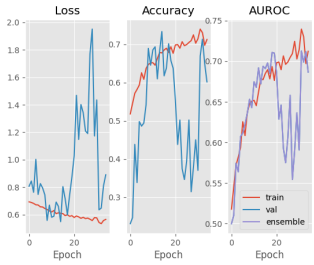
The data was acquired from TCGA repository at `https://portal.gdc.cancer.gov/repository`

B Feature Extrator Results

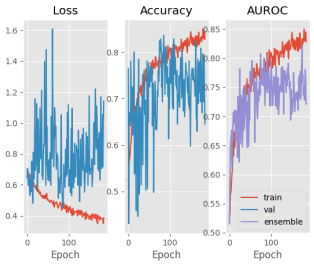
B.1 ResNet50



(a) Fold 7/10



(b) Fold 9/10

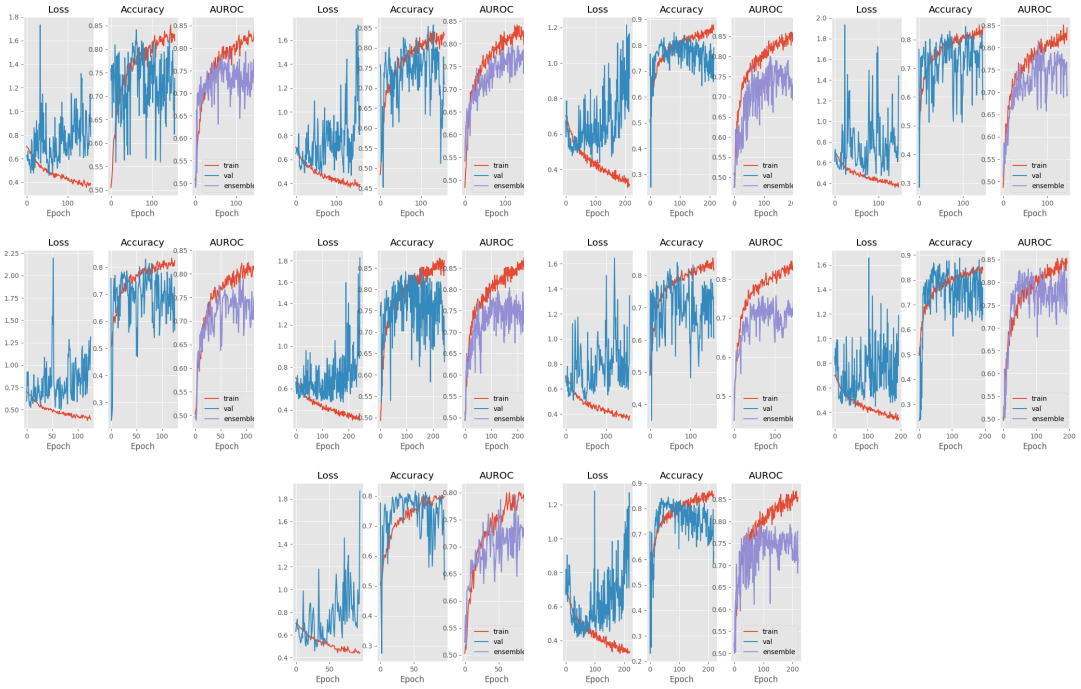


(c) Fold 10/10

Fold	1	2	3	4	5	6	7	8	9	10
AUC	0.789	0.773	0.793	0.840	0.674	0.693	0.795	0.853	0.713	0.821

Table 6: Fold Validation

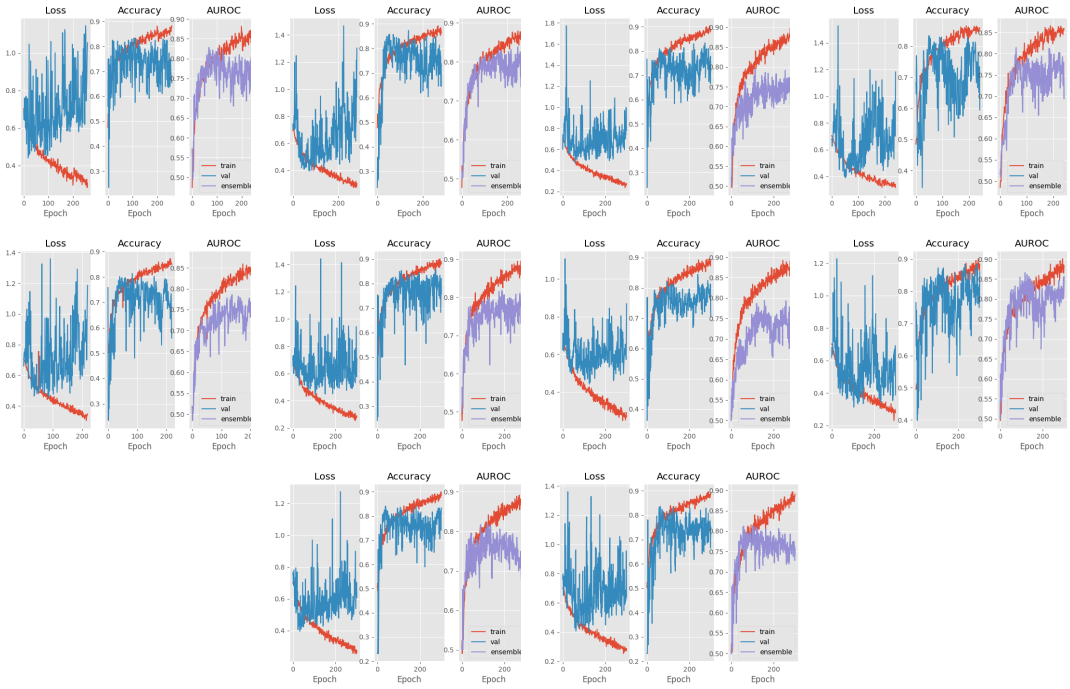
B.2 ResNeXt50



Fold	1	2	3	4	5	6	7	8	9	10
AUC	0.794	0.795	0.801	0.815	0.786	0.804	0.760	0.838	0.795	0.793

Table 7: Fold Validation

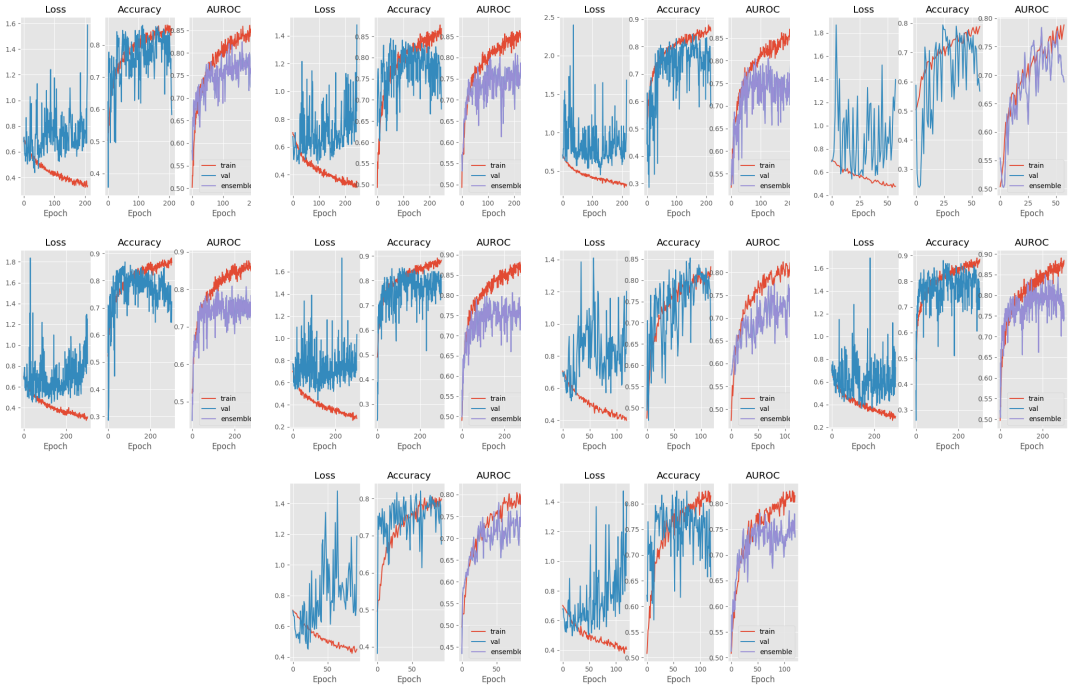
B.3 ResNeXt50 w/ SE



Fold	1	2	3	4	5	6	7	8	9	10
AUC	0.823	0.843	0.796	0.814	0.786	0.818	0.794	0.866	0.820	0.813

Table 8: Fold Validation

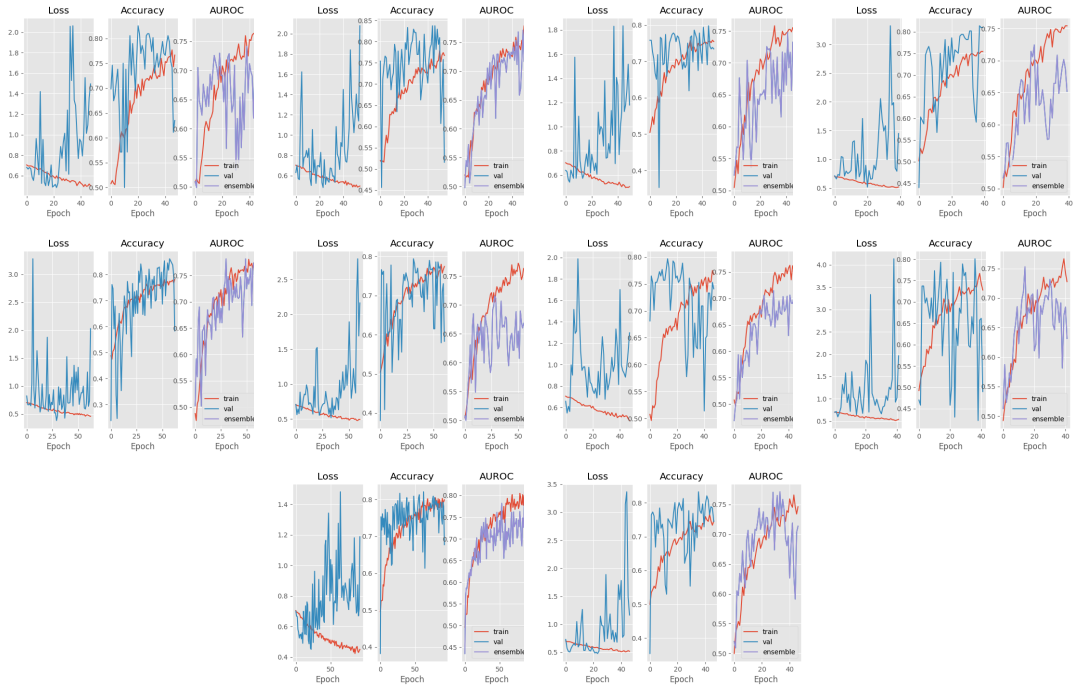
B.4 ResNet101



Fold	1	2	3	4	5	6	7	8	9	10
AUC	0.823	0.810	0.806	0.784	0.808	0.806	0.775	0.850	0.808	0.786

Table 9: Fold Validation

B.5 ResNeXt101 w/ SE



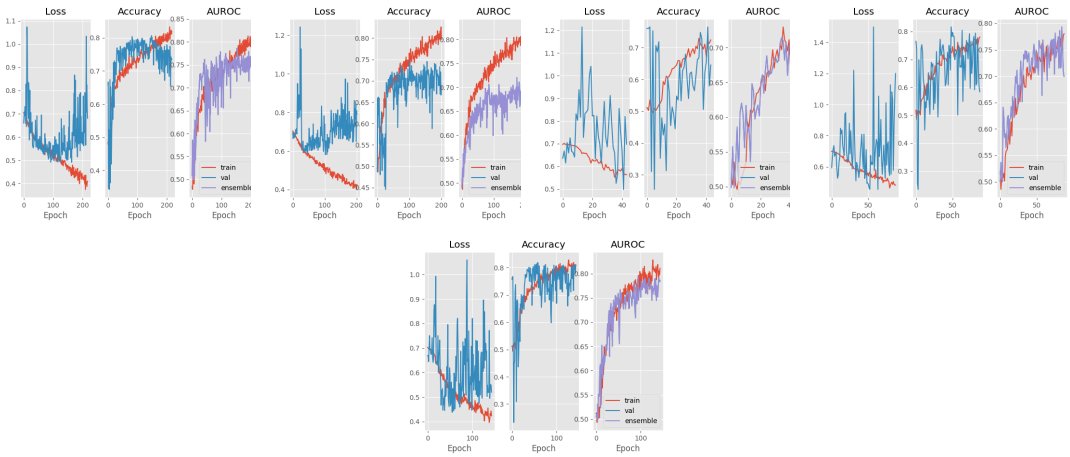
Fold	1	2	3	4	5	6	7	8	9	10
AUC	0.742	0.771	0.748	0.725	0.7815	0.716	0.719	0.753	0.736	0.771

Table 10: Fold Validation

C Hormone Receptor Results

C.1 Estrogen Receptor

C.1.1 Cross-Validation



Fold	1	2	3	4	5
AUC	0.776	0.800	0.772	0.793	0.802

Table 11: Fold Validation

C.1.2 Test Dataset

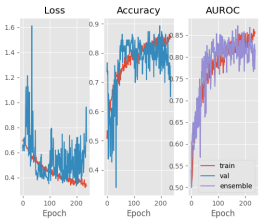
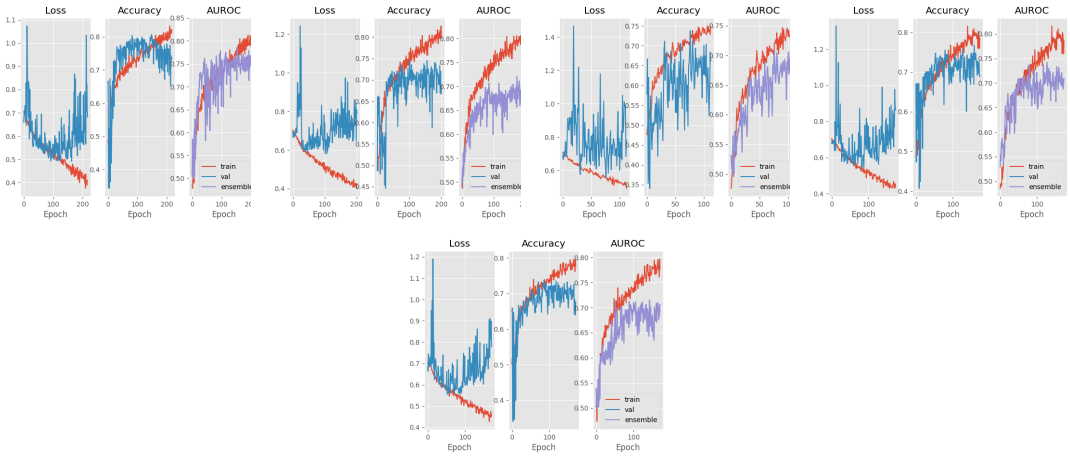


Figure 28: ER Training for Test

C.2 Progesterone Receptor

C.2.1 Cross-Validation



Fold	1	2	3	4	5
AUC	0.787	0.720	0.709	0.741	0.715

Table 12: Fold Validation

C.2.2 Test Dataset

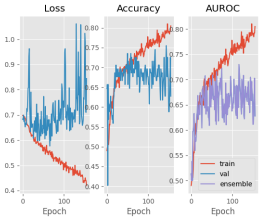
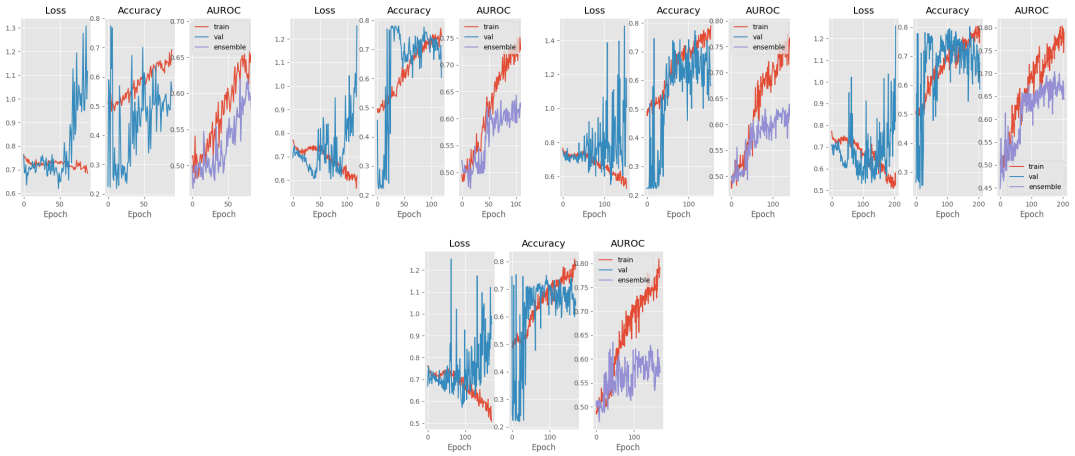


Figure 30: PR Training for Test

C.3 Human Epidural Growth Factor Receptor 2 (HER2)

C.3.1 Cross-Validation



Fold	1	2	3	4	5
AUC	0.620	0.653	0.640	0.704	0.635

Table 13: Fold Validation

C.3.2 Test Dataset

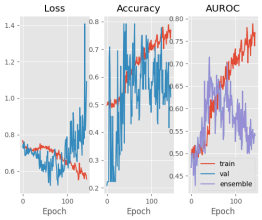
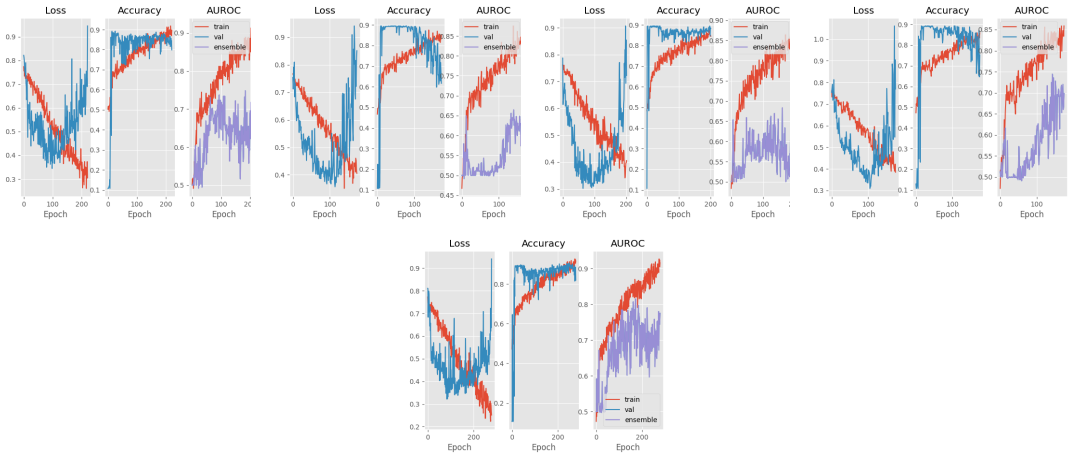


Figure 32: HER2 Training for Test

C.4 Triple Negative (TNBC)

C.4.1 Cross-Validation



Fold	1	2	3	4	5
AUC	0.748	0.665	0.683	0.744	0.815

Table 14: Fold Validation

C.4.2 Test Dataset

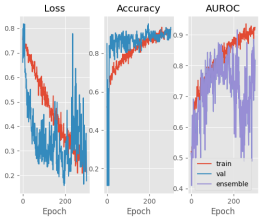


Figure 34: TNBC Training for Test