Student: Edwin Meyers
Date: 14-Sep-20

Machine Learning Capstone Project

Prostate cANcer graDe Assessment (PANDA) Kaggle Challenge

Prostate cancer diagnosis using the Gleason grading system with neural networks

I. Definition

1. Project Overview

The objective of the project was to develop a digital method to determine the ISUP score (score that indicates the presence and stage of prostate cancer) of a given prostate tissue biopsy. The project was based on the Kaggle competition named “Prostate cANcer graDe Assessment (PANDA) Kaggle Challenge” (see competition overview here). Among other resources, the challenge page contains 10,616 biopsy images, a similar number of label masks for said biopsy images, and the file key “train.csv” which contains information on each biopsy image, including the image id (same as file name of corresponding images), ISUP score, Gleason score (other cancer-related indicator for prostate biopsies), and the name of the hospital that provided the biopsy image (Karolinska and Radboud were the two hospitals that participated in the preparation of this competition).

2. Problem Statement

Over 1.2 million new cases of prostate cancer (PCa) are reported each year.[1] PCa represents 7% of all new cancer cases in men.[2] The key to lowering the number of PCa deaths is developing more precise diagnostics. Diagnosis of PCa is based on the grading of prostate tissue biopsies. These tissue samples are examined by a pathologist and scored according to the Gleason grading system. Diagnosing PCa, however, can sometimes be more of an art than an actual science. Two given pathologists may grade a given tissue sample differently, which could have potentially life-impacting effects on patient treatments and outcomes.

In more project-based terms, this is a classification problem where a model had to be trained with augmented versions of a database of images in order for it to accurately predict the ISUP score of a prostate biopsy image.

3. Metrics

The two main metrics used to determine the success of the project were 1) the validation and accuracy rates of the trained models and 2) the quadratic weighted kappa submission score generated by the competition which measures the agreement between two outcomes. Since the hidden test pictures have been evaluated by multiple pathologists, this metric can vary from 0 (random agreement) to 1 (complete agreement). In the event that there is less agreement than expected, the metric may go below 0. More details on how the quadratic weighted kappa is calculated is available in the Evaluation section of the Kaggle challenge webpage.

[1] US National Library of Medicine and National Institute of Health (link)
[2] US National Library of Medicine and National Institute of Health (link)

II. Analysis

1. Data Exploration

Files and folders in dataset with respective descriptions:

- Folder: train_images/
  Comprised of 10,616 multi-level .TIFF files. This was somewhat difficult to manipulate at first. Ultimately, of the three levels available (high resolution, medium resolution, and low resolution), the second one was chosen.

- Folder: train_label_masks/
  Comprised of 10,516 single-level .TIFF files that serve as masks to corresponding biopsy images in the "train_images/" folder. The masks have either 3 or 6 posible label scores depending on the data provider, each score detailed below:

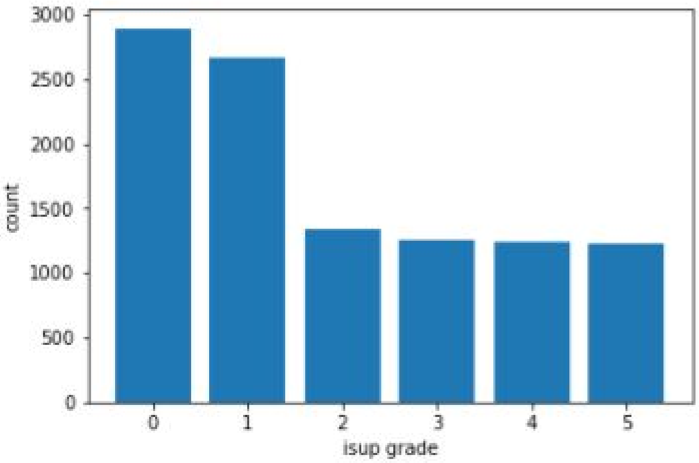
  Radboud: Prostate glands are individually labelled. Valid values are:
  0: background (non tissue) or unknown
  1: stroma (connective tissue, non-epithelium tissue)
  2: healthy (benign) epithelium
  3: cancerous epithelium (Gleason 3)
  4: cancerous epithelium (Gleason 4)
  5: cancerous epithelium (Gleason 5)

  Karolinska: Regions are labelled. Valid values are:
  0: background (non tissue) or unknown
  1: benign tissue (stroma and epithelium combined)
  2: cancerous tissue (stroma and epithelium combined)

- File: train.csv
  File key for all images provided. Specifically, information provided is the image id (same as the file name of the corresponding image), isup grade, gleason score, and data provider

- Files: submission.csv and test.csv
  A valid submission file. This is a notebooks-only competition; the downloadable test.csv and sample_submission.csv have been truncated. The full [hidden] versions are only available to the submitted notebooks.

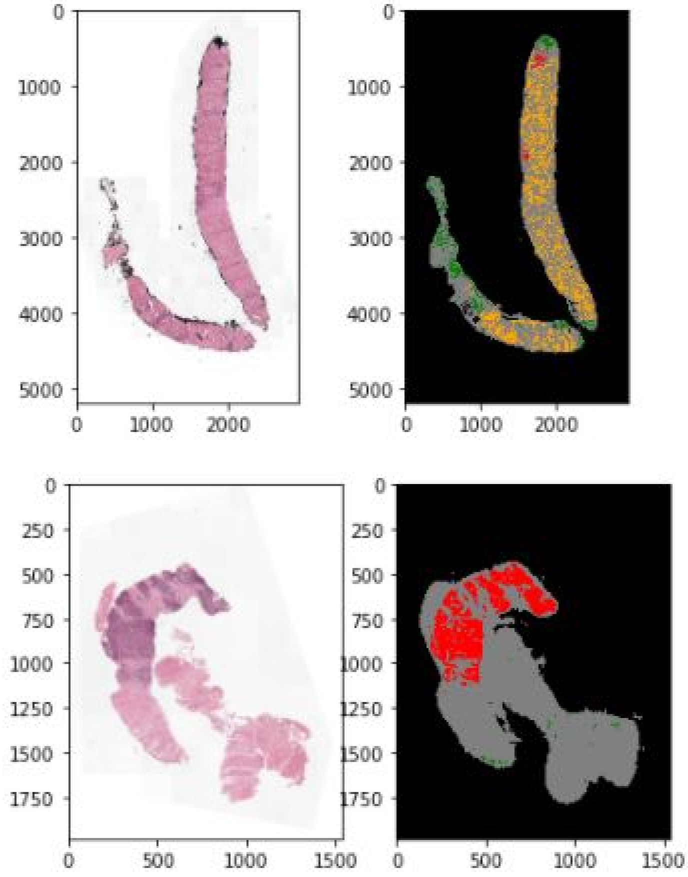
As can be seen in the chart below, the training data has significantly more samples of lower score isup grades (0 and 1, relating to healthy and low risk, respectively). This could be an acceptable skew in the data because in the "real world", most people do not have prostate cancer.



2. Exploration Visualization

Of the 10,616 biopsy images, all but 100 have a corresponding label map (see previous subsection “Data Exploration” for description of label images). Visualizing the label maps required assigning a color to the different values using the .cmap command from matplotlib.

Biopsy image (left) next to label maps (right). See section above for information on label maps.



On average, it was visually estimated that tissue samples take up between 5% and 20% of an image’s surface area.The rest consists of either blank background or section delimitations traced with markers by pathologists. This means that ~80-95% of a biopsy image is noise. As such, image augmentations were needed to increase the model's ability to adequately train. Please refer to Section III. Methodology, subsection Data Preprocessing for information on how images were augmented.

3. Algorithms and Techniques

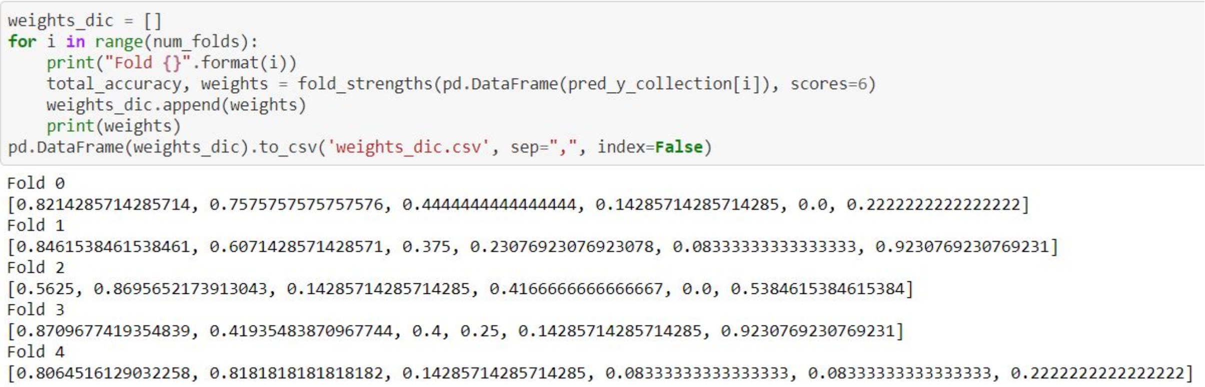
Algorithms and techniques that were explored or used in this project were the following:

- ***Image augmentation / preprocessing:*** Used the tile method to augment image quality and reduce image noise (discussed in greater detail in Section III, Data Preprocessing)
- ***Modifying PyTorch DataLoader:*** Modified PyTorch DataLoader to:
  - Load files that are specified in a .CSV file instead of only being able to load all files in a given directory by default (can be seen in all notebooks in cells where the class "load_csv(Dataset)" is defined, except for the image augmentations notebook).
  - Manipulate input image data if required (eg. when submitting the model to Kaggle, image augmentations were performed directly in the DataLoader modifier. Please see any notebook in repository with "Analysis and Submission" in the file name, in the cells where the class "load_csv(Dataset)" is defined)
  - Transform labels into vectors (aka binning the labels, see sub-subsección "BCEWithLogitsLoss Criterion" further below for more context)
- ***Training fully connected NN's:*** Trained fully connected neural networks with augmented images. Originally used EfficientNet-B4 but finished using the lighter EfficientNet-B0 in the end to reduce training times and experiment with different prediction/training methods
- ***K-Folds method:*** Trained models using k-folds method. This method did not generate superior results but the author is willing to admit that the implementation could have been improved, potentially with better data (data quality was only improved in the last iterations of the project and did not include k-folds method as part of such iterations), lighter models that could have been trained for longer, or a different method to calculating the average of the predictions of the k different models.

  When it came to making a final prediction using as inputs the different predictions generated by each k-fold model, two methods were used:

  (i) calculating the simple average and

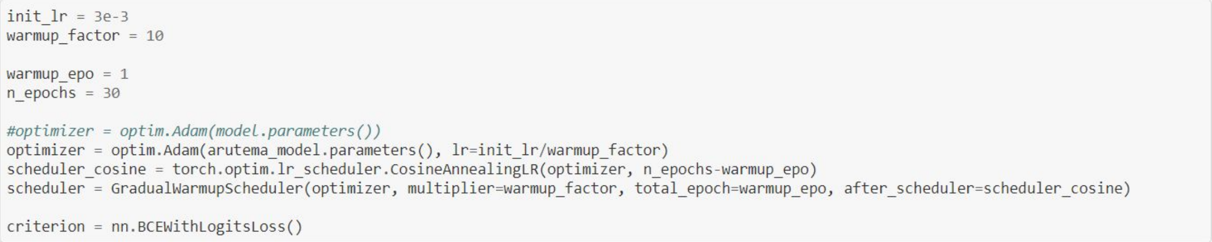
  (ii) calculating the weighted average, where the weights corresponded to the accuracy that each k model had when predicting different isup scores correctly. For example,using the accuracy matrix from the code output below, if CNN's from folds 0, 1, 2, 3, 4 predicted isup scores 3, 5, 2, 2, 1, respectively, then the weights assigned to each of those predictions would be 0.14, 0.92, 0.14, 0.40, 0.82, respectively.



```
weights_dic = []
for i in range(num_folds):
    print("Fold {}".format(i))
    total_accuracy, weights = fold_strengths(pd.DataFrame(pred_y_collection[i]), scores=6)
    weights_dic.append(weights)
    print(weights)
pd.DataFrame(weights_dic).to_csv('weights_dic.csv', sep=",", index=False)
```

```
Fold 0
[0.8214285714285714, 0.7575757575757576, 0.4444444444444444, 0.14285714285714285, 0.0, 0.2222222222222222]
Fold 1
[0.8461538461538461, 0.6071428571428571, 0.375, 0.23076923076923078, 0.08333333333333333, 0.9230769230769231]
Fold 2
[0.5625, 0.8695652173913043, 0.14285714285714285, 0.4166666666666667, 0.0, 0.5384615384615384]
Fold 3
[0.8709677419354839, 0.41935483870967744, 0.4, 0.25, 0.14285714285714285, 0.9230769230769231]
Fold 4
[0.8064516129032258, 0.8181818181818182, 0.14285714285714285, 0.08333333333333333, 0.08333333333333333, 0.2222222222222222]
```

- ***Learning rate scheduler:*** In later iterations of the project, the author used a learning rate scheduler to gradually decrease the learning rate across 30 epochs from 3e-4 to 9e-6, which, as can be seen in training outputs from notebook 8a_base-model-wlogits-retrained-arutema-s-method.ipynb, likely helped ensure that the training and validation losses continued to decrease with each epoch.



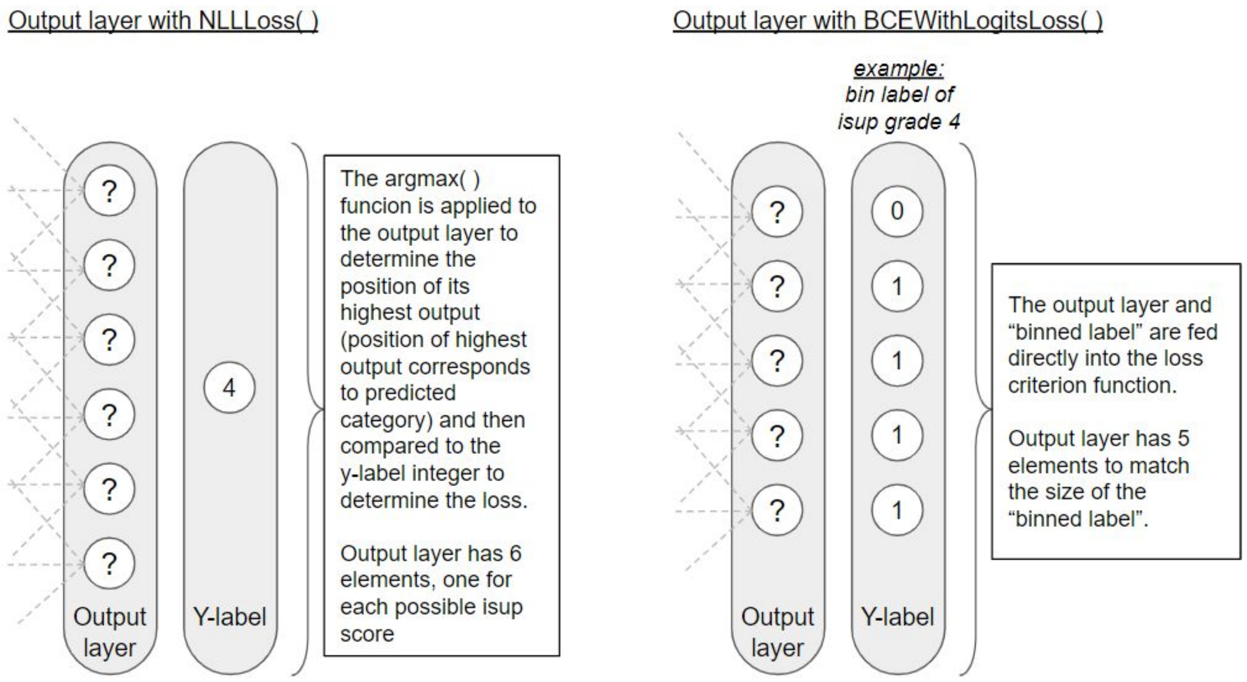
```
init_lr = 3e-3
warmup_factor = 10

warmup_epo = 1
n_epochs = 30

#optimizer = optim.Adam(model.parameters())
optimizer = optim.Adam(arutema_model.parameters(), lr=init_lr/warmup_factor)
scheduler_cosine = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, n_epochs-warmup_epo)
scheduler = GradualWarmupScheduler(optimizer, multiplier=warmup_factor, total_epoch=warmup_epo, after_scheduler=scheduler_cosine)

criterion = nn.BCEWithLogitsLoss()
```

- **Arutema method:** The process of using a previously trained model to determine images in the dataset that (i) were predicted incorrectly beyond a certain error threshold and (ii) likely caused “noise” in the training process, ultimately affecting the model’s prediction accuracy. Please refer to section Section III. Data Preprocessing for more information.
- **BCEWithLogitsLoss Criterion:** The model was originally being trained with a NLLLoss criterion. This initial election was made because the author had created a classification model before using the NLLLoss criterion with success (see image classifying project in question here). However, upon exploring other loss criterions, BCEWithLogitsLoss seemed to be the most adequate. NLLLoss was used for training a model that had to classify discrete objects that have very distinct features from one another (eg. flower classifier, dog classifier, object classifier) whereas the problem at hand required determining where a tissue sample sits on a continuum of potential isup scores. For illustrative purposes, this difference can be compared to the difference between training a model to classify animals versus creating a model that can determine the age of a given breed of dogs.

  NLLLoss( ) criterion required using the argmax( ) function in order to determine the position of the highest value in the model’s output layer, which in turn was used as the model’s predicted isup grade. Then, the loss criterion was calculated between the predicted isup grade and the label.

  BCEWithLogitsLoss( ) criterion required “binning the labels” as shown below:

  isup_grade = 0 ⇔ label = [0,0,0,0,0]
  isup_grade = 3 ⇔ label = [1,1,1,0,0]
  isup_grade = 5 ⇔ label = [1,1,1,1,1]


Output layer with NLLLoss( )
?
?
?
?
?
?
Output layer
4
Y-label
Output layer with BCEWithLogitsLoss( )
example: bin label of isup grade 4
?
?
?
?
?
Output layer
0
1
1
1
1
Y-label

The argmax( ) funcion is applied to the output layer to determine the position of its highest output (position of highest output corresponds to predicted category) and then compared to the y-label integer to determine the loss.

Output layer has 6 elements, one for each possible isup score

The output layer and "binned label" are fed directly into the loss criterion function.

Output layer has 5 elements to match the size of the "binned label".

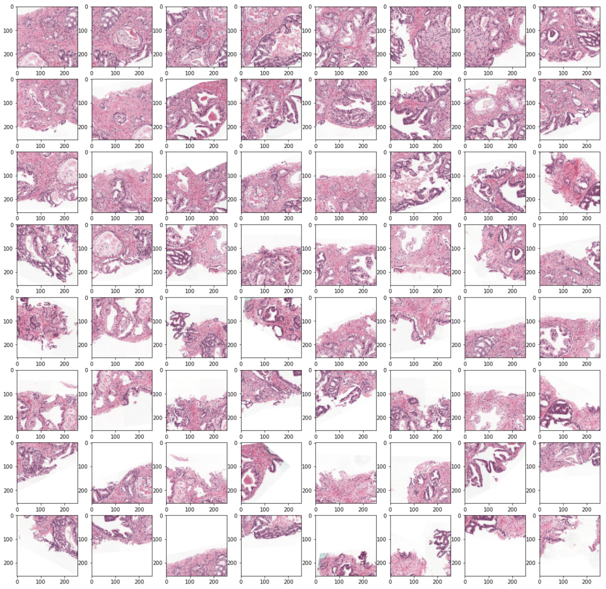
Note: "?" represents output values. A LogSoftmax( ) layer is applied to outputs used as inputs for the NLLLoss( ) criterion. BCEWithLogitsLoss( ) already has a built-in sigmoid layer, allowing linear output to be used directly as criterion input.

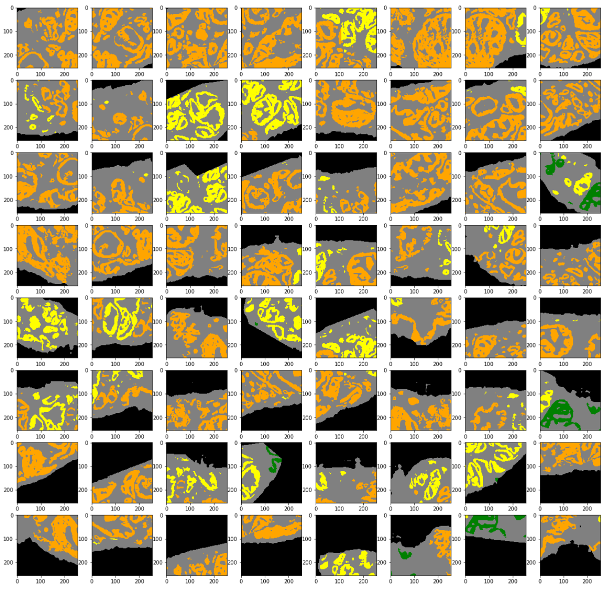
Using the BCEWithLogitsLoss( ) criterion helped increase the model's prediction accuracy, as described in Section III. Methodology, subsection Refinement. Implementing this new criterion and understanding how it implicitly communicated to the model a sense of continuity among the output categories was a valuable learning experience.

Methods explored but never implemented due to lack of GPU time and doubts on meaningful value add over simply using the original tile method (see first point of this subsection):

- **Making better use of label masks:** There didn't seem to be any teams making good use of the label masks. The author's intention was to use the tile method (see first point in this subsection) to later save each tile as an individual image. The individual tiles could be assigned respective Gleason scores using scores provided in the label mask images. Images would then be saved individually and would contain information on the biopsy image they were derived from and the gleason score of the individual smaller image. Later, after training a model with 100,000's of smaller images, isup scores can be predicted for biopsy images by determining the Gleason scores of smaller portions of the tissue sample.

  Tile method applied to biopsy image (left) and same regions mapped onto label mask images (right)





4. Benchmark

There are three benchmarks that were used by the author:

(i) 16.7% accuracy is the first benchmark because it is the accuracy rate that corresponds to complete randomness. Since there are 6 different ISUP scores, one could randomly categorize biopsy images and on average achieve a 16.7% accuracy rate.

(ii) Accuracy rates of previous models to measure performance improvements.

(iii) Kaggle submission score of 71.9: This Capstone project is based on a Kaggle competition. Kaggle has a leaderboard that displays the score of other participants' models (please find the leaderboard here). There are over 4,000 submissions listed in the leaderboard. On average, submissions achieved an evaluation score of 71.9 and a maximum score of 93.4 (see "Section 6. Evaluation metrics" for more information on significance of metric). Given that this is the author's first Kaggle competition, he will be happy with a submission score that is above that of half of the registered submissions.

III. Methodology

1. Data Preprocessing

Data preprocessing consisted of two main methods: (i) using the tile method to reduce noise in the biopsy images and (ii) applying the Arutema method to remove noise-generating images in the dataset. Please find descriptions for both methods here below:

(i) Tile Method:
Consists of reducing the noise caused by background color (or any part of the image that isn't tissue matter, such as markers)

Step 1: An image is divided into a grid. Each grid is referred to as a square or a region.

Step 2: Squares are ranked according to the percentage of white (or off-white colors) colored pixels in the squares. White pixel presence is generally indicative of background color, which often generates noise.

Step 3: Sorted regions are “glued” together to create a final, augmented image.

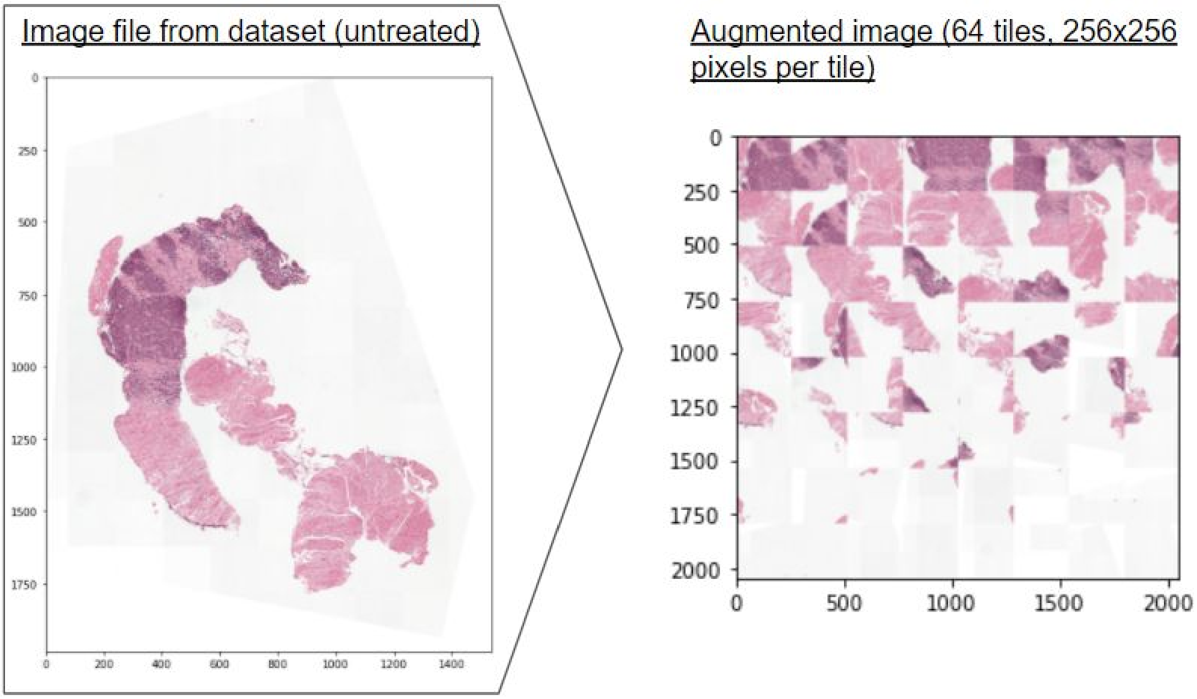
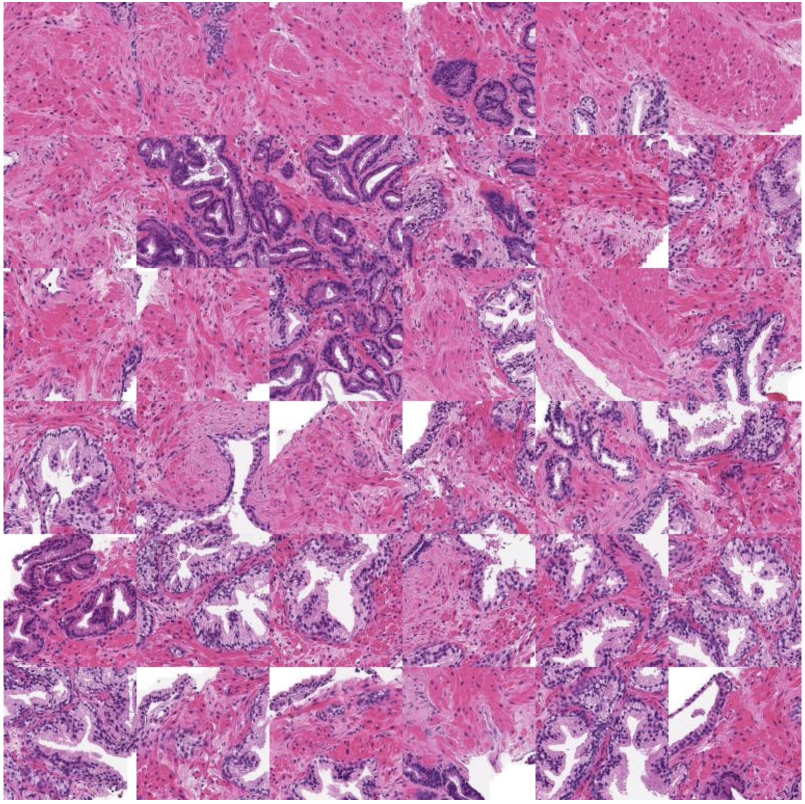


Image file from dataset (untreated)

Augmented image (64 tiles, 256x256 pixels per tile)

The final model was trained using 36 tiles, 128x128 pre-augmented images. A sample image is displayed below.

36x128x128 pre-augmented image



(ii) Arutema Method:

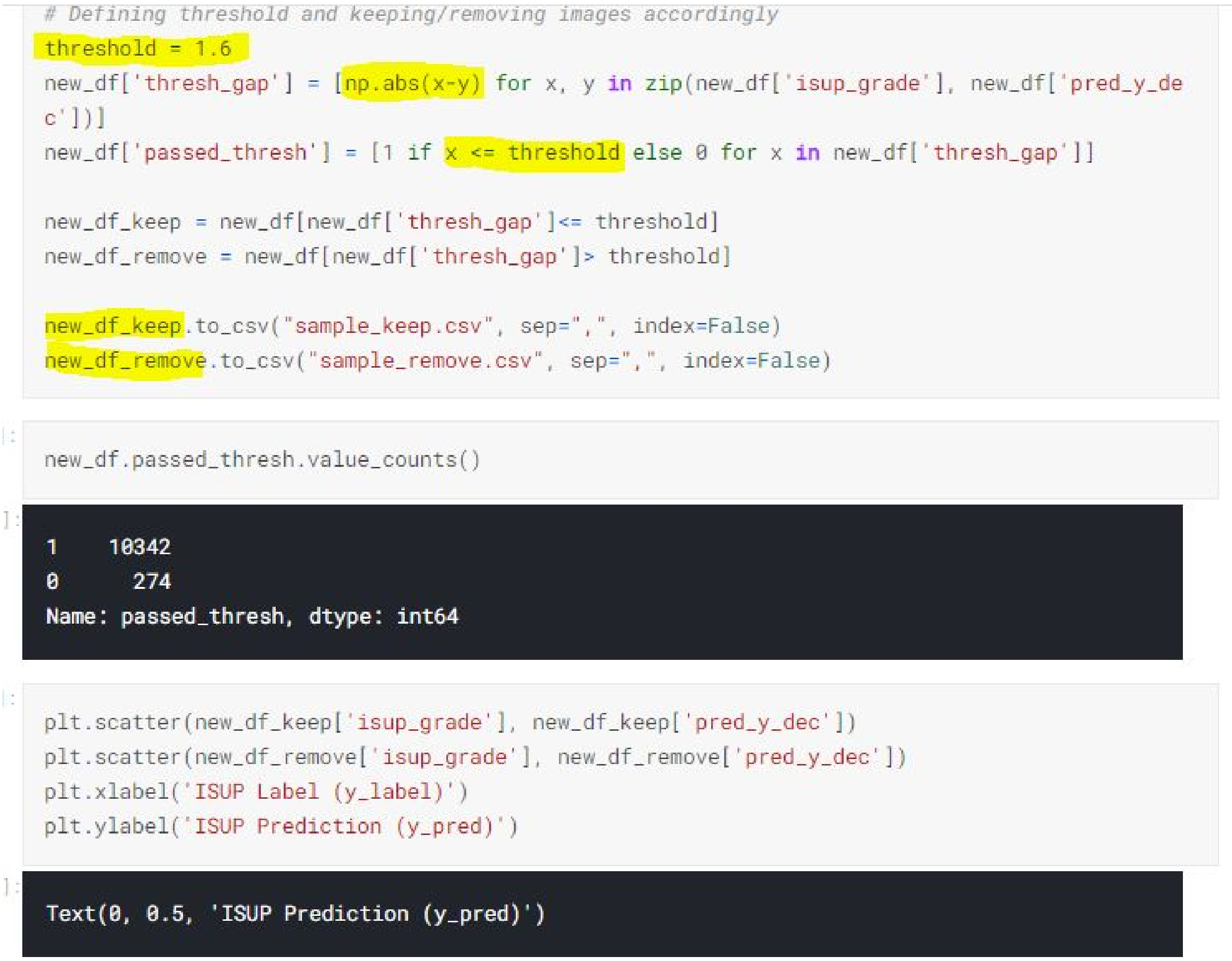
Consists of filtering “noisy” images from the dataset according to how well a previously trained model was able to predict their ISUP scores of images within a certain threshold. Please note that this methodology could only be used once the BCEWithLogitsLoss( ) criterion was implemented in the model.

Step 1: Train a logits-based / BCEWithLogitsLoss model on the given dataset
Step 2: Have the model generate predictions for all items in the dataset. Predictions should be calculated by summing the sigmoid values of the output layer, without rounding the result.
Step 3: Define a threshold for error gaps. For example, if a given non-rounded ISUP prediction is 3.4 and the label is 5, then the error gap, calculated using the function `error_gap = np.abs(prediction - label)`, would be 1.6. The gap determines the images ability to accurately predict an image and the threshold determines the user’s tolerance for inaccuracies.
Step 4: Train a new model using only images that the previous model was able to predict within the defined error threshold (i.e., new_df_keep, later saved as sample_keep.csv in the code snippet below)

This preprocessing step was very powerful. It achieved an astonishing 97.9% validation accuracy rate and a 95.7% test validation rate (see snippet of Arutema algorithm being applied to create new dataset keys,, sample_keep.csv and sample_remove.csv).

This step was a highly enlightening experience because it shows how the model needs to be trained with clear and concise examples that are easy to interpret and are likely more representative of the ground truth. That way, the model knows which key features to look for, even in new cases where the features may be more difficult to ascertain than others.

Applying the Arutema method helped increase the final submission score from a previous high of 61.0 submission score to 76.6, representing a +15.5 point increase! As can be seen in the cell code below, the significant score increase was achieved by removing only 2.6% of the images in the database (274 of 10,616 images)[3].



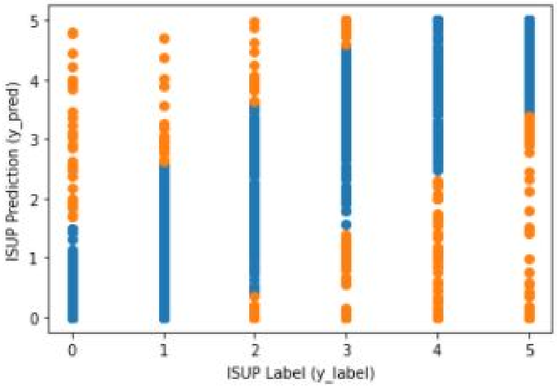
```
# Defining threshold and keeping/removing images accordingly
threshold = 1.6
new_df['thresh_gap'] = [np.abs(x-y) for x, y in zip(new_df['isup_grade'], new_df['pred_y_dec'])]
new_df['passed_thresh'] = [1 if x <= threshold else 0 for x in new_df['thresh_gap']]

new_df_keep = new_df[new_df['thresh_gap']<= threshold]
new_df_remove = new_df[new_df['thresh_gap']> threshold]

new_df_keep.to_csv("sample_keep.csv", sep=",", index=False)
new_df_remove.to_csv("sample_remove.csv", sep=",", index=False)
```

```
new_df.passed_thresh.value_counts()
```

```
1    10342
0      274
Name: passed_thresh, dtype: int64
```

```
plt.scatter(new_df_keep['isup_grade'], new_df_keep['pred_y_dec'])
plt.scatter(new_df_remove['isup_grade'], new_df_remove['pred_y_dec'])
plt.xlabel('ISUP Label (y_label)')
plt.ylabel('ISUP Prediction (y_pred)')
```

```
Text(0, 0.5, 'ISUP Prediction (y_pred)')
```



[3] Note: the final score was 79.0 by training the models with the larger 36x128x128 dataset

2. Implementation and Refinement

Implementation and refinement steps are described below in the same order in which they occurred.

Description of implementation process

Step 1 - Setting up

- *The project was first being built on AWS but, because of GPU costs and dataset management complications, the project was eventually moved to Kaggle which offers free GPU-enabled kernels and easy access to a variety of data in addition to the competition data.*
- *The pictures were multi-level “.TIFF” files, each level containing very different sized images. Of the three levels available, the middle one was eventually chosen because associated image sizes contained enough detail all while not being so big that they would overload the GPU’s memory.*

Step 2 - Training and predicting with the first running model

> Notebook location: main repository
> Notebook name:

- *2_first_model_draft.ipynb*

> Summary:

The objective was to set up an initial model that would serve as a foundation for further features to be built upon. The “foundation” model was largely borrowed from another notebook made by the author for a previous Udacity project (see image classifier here). Once built, the accuracy was measured (28%, not much better than random state = 1/(6 isup categories) = 16.7% accuracy) and next steps were defined according to quick calculations / time measurement / draft work done in the notebook.

> Principal achievements:

- *Understanding how to open a multi-level “.TIFF” file and determining which level to use for the rest of the project.*
- *Modifying the dataloader’s “__get_item__” class definition so that the dataloader can take excel sheets of specified image id’s and labels instead of only loading what is in a specified folder. This was very useful given that there were 10,616 images and having the flexibility to sift through images nimbly was important (especially, as the reader will later see, for the Arutema method).*
- *Creating a working model*

It was clear that next steps had to be taken to increase the quality of the data inputs to hopefully improve the quality of the predictions.

Step 3 - Image augmentations

> Notebook location: Main repository
> Notebook name:

- *3_image-augmentations-masks-and-label-mapping.ipynb*

> Summary of step:

From what the author could measure for a small sample of images, biopsy tissues usually only account for less than 5% of the image surface. The rest is background/glass laminate. This means that 95% of the image is essentially noise. The author borrowed algorithms from Kaggle user PAB97 (see notebook here) to implement image augmentations into the data refining process.

> Principal achievements:

- *Learning basic image manipulation methods and how to transform a biopsy tissue into a with a high tissue-to-background concentration, as described in Section III. Methodology, Subsection 1. Data Preprocessing.*
- *Learning how to open masks and map mask labels to image enhancements, as described in Section II. Analysis, Subsection 2. Exploration Visualization.*

Step 4 - Base model and first submission (57.2 submission score)

> Notebook location: Main repository
> Notebook names:

- *4a_base-model-no-folds.ipynb*
- *4b_base-model-analysis-and-submission.ipynb*

> Summary of results:

- *56.6% accuracy rate with validation dataset*
- *57.2 Kaggle submission score (see how score is calculated in the document named "proposal.pdf")*

> Model/Training information:

- *10 epochs*
- *Model = Efficientnet-B4*
- *criterion = nn.NLLLoss()*
- *optimizer =* optim.Adam(model.parameters(), lr=0.001)

> Principal achievements:

- *Implementing image augmentation for model training and increasing accuracy from 28% to 57%*
- *Figuring out how to submit a model to the Kaggle competition. This was a very complicated part of the competitio*n.

Step 5 - K-folds training and analysis

> Notebook location: Main repository
> Notebook names:

- *5a_Training model with k-folds.ipynb*
- *5b_K-Folds model - Analysis and submission.ipynb*

> Summary of results

- *Please see complete comments on challenges and methods explored with k-folds in Section II. Analysis, subsection 3. Algorithms and techniques*
- *11% accuracy rate with validation dataset*
- *Not submitted to Kaggle due to low score.*

> Model/Training information:

- *10 epochs*

- *5 folds*
- *Efficientnet-B4*
- *criterion = nn.NLLLoss()*
- *optimizer = optim.Adam(model.parameters(), lr=0.001)*

> Principal achievements:

- *Learning how to implement k-folds model training method*
- *Learn new weights determination and manipulation technique (eg. what weights to assign to different models' outputs depending on their historical accuracy rates with different predicted isup grades)*
- *Learning how to save and load checkpoints for all folds*

Individual k-models reached accuracy rates as high as 59% (please ignore the accuracy rate's formatting error in the notebook), but the author was unable to find appropriate weights that would allow a weighted (or unweighted) average to yield more accurate results than could any single model

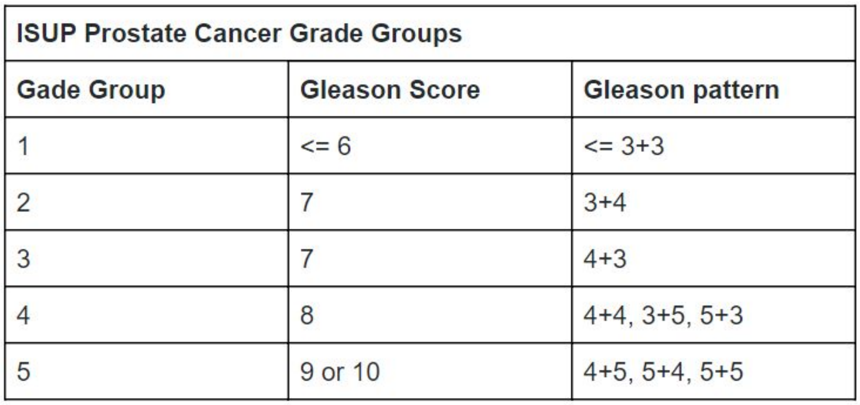
Step 6 - Gleason scaled model

> Notebook location: Main repository
> Notebook name:

- *6_base_model_no folds_10_g_score_learning.ipynb*

> Summary of Step:

- *The objective of this notebook was to see if accuracy rates would increase by increasing the number of output classes (defined by number of possible Gleason scores, 10 classes) with the hope that it would increase the accuracy when mapping Gleason scores to ISUP scores (6 classes). The reason why this was developed was because the original model was having difficulty accurately predicting biopsies with ISUP scores 4 and 5. These two ISUP scores, however, can be mapped to 5 different Gleason scores (see image below).*



ISUP Prostate Cancer Grade Groups		
Gade Group	**Gleason Score**	**Gleason pattern**
1	<= 6	<= 3+3
2	7	3+4
3	7	4+3
4	8	4+4, 3+5, 5+3
5	9 or 10	4+5, 5+4, 5+5

> Summary of results:

- *42.7% accuracy rate with validation dataset*
- *Not submitted to Kaggle due to low score*

> Model / Training information:

- *10 epochs*
- *Model = Efficientnet-B4*
- *criterion = nn.NLLLoss()*
- *optimizer = optim.Adam(model.parameters(), lr=0.001)*

> Principal achievement:

- *Learning that mapping based efficiencies may not necessarily increase accuracy rates. It is safe to assume that the model implicitly takes care of these types of category mappings.*

Step 7 - Logits base model and submission (61.0 submission score)

> Notebook location: *Main repository*
> Notebook names:

- *7a_base_model_no_folds_Logits.ipynb*
- *7b_base_model_w_logits_-_analysis_ and_submission.ipynb*

> Summary of results:

- *59.2% accuracy rate with validation dataset*
- *61.0 Kaggle submission score (see how score is calculated in the document named "proposal.pdf")*

> Model / Training information:

- *30 epochs*
- *Model = Efficientnet-B0*
- *criterion = nn.BCEWithLogitsLoss()*
- *scheduler was cosine with gradual warm up*
- *optimizer = optim.Adam(model.parameters(), lr=learning scheduler)*

> Principal achievement:

- *Implementing new criterion (BCEWithLogitsLoss) and understanding how it implicitly communicates to the model a sense of continuity in the output categories (tissue biopsy samples have varying but continuous grades of cancer vs. classifying different types of animals is a discrete type of classification)*
- *Implementing a learning scheduler*
- *Using Efficientnet-B0, which is much lighter and faster to train than Efficientnet-B4, which allowed the author to train over many more epochs and experiment with learning rate schedulers.*

Step 8 - Arutema method with logits base model and submission (76.6 submission score)

> Notebook location: *Main repository*
> Notebook names:

- *8a_base_model_wLogits_retrained_Arutema's method.ipynb*;
- *8b_Arutema base model w logits - analysis and submiss.ipynb*

> Summary of step:

- *Method described in Section III. Methodology, subsection 1. Data preprocessing, sub-subseccion (ii) Arutema Method*

> Summary of results:

- *97.9% accuracy rate with validation dataset*
- *76.6 Kaggle submission public score, above the 71.9 average competition submission score (see how Kaggle competition score is calculated in the document named "proposal.pdf")*

> Model / Training information:

- *30 epochs*
- *Model = Efficientnet-B0*
- *criterion = nn.BCEWithLogitsLoss()*
- *scheduler was cosine with gradual warm up*
- *optimizer = optim.Adam(model.parameters(), lr=learning scheduler)*

> Principal achievement:

- *Understanding and seeing first hand the power of denoising data through the Arutema method. In the end, removing 3% of the dataset resulted in a +16 point increase in the final submission score! Less can certainly mean more when it comes to data wrangling / filtering.*
- *Finishing the project knowing that if there were a few more days left, it would be very possible to achieve a significantly higher score by creating a 64x(256x256) pre-tiled image dataset, train new model with the dataset using methods described in this section, and submitting the model for grading.*

Step 9 - Arutema method with logits base model and submission with 36x128x128 images (79.0 submission score)

> Notebook location: *Main repository*
> Notebook names:

- *9a_base-model-no-folds-logits-36x128x128-level1*;
- *9b_arutema-base-model-w-logits-a-s-36x128x128 (link to notebook with score here)*

The model had the same characteristics as the one described in step 8, only that the model training and predicting was performed with images augmented into 36x128x128 glued images, which resulted in the higher submission score.

IV. Results

1. Model Evaluation and Validation

The final model is described in Section III. Methodology, subsection 2. Implementation and Refinement, Step 9.

- Model parameters and training evaluation:

- 30 epochs: With each additional epoch, the train loss continued dropping and the validation accuracy generally rose, especially with the learning scheduler reducing the learning rate gradually. Less epochs would have likely translated to lower accuracy scores. Unfortunately, due to GPU time restraints, the author couldn't explore the possibilities of training with more epochs.
- Model = Efficientnet-B0: As mentioned before, the model was first trained with the heavier pre-trained model Efficientnet-B4. The model took up a lot of memory and was slower to train (up to 10 epochs before reaching Kaggle's GPU time limit) and more difficult to experiment with. B0 was preferred for the aforementioned disadvantages of B4.
- criterion = nn.BCEWithLogitsLoss(): Details regarding results obtained with models using NLLLoss and BCEWithLogitsLoss criterions are described in Section III. Methodology, subsection 2. Implementation and Refinement, Steps 4 and 7, respectively. 57.2 vs. 61.0
- Scheduled learning rate: With each new training epoch, the learning rate decreased, from a maximum of 0.003 to eventually reaching 0.000009. With each decrease in the learning rate, the training loss decreased too. This was not the case before implementing the scheduled learning rate, as can be seen in the output of notebooks 4a_base-model-no-folds.ipynb and 5a_training-model-with-k-folds.ipynb from the main repository.
- optimizer = optim.Adam(model.parameters(), lr=learning scheduler): the Adam optimizer helps avoid local minimas and automatically associates a momentum rate to a provided learning rate to help models get over local minimas.
- 36x128x128 augmented image dimensions: Ideally, the model would have been trained on 64x256x256x64 augmented images. Due to limited GPU and project time, the final model was instead trained on a pre-prepared database of 36x128x128 augmented images.

16.7% accuracy is an initial benchmark used to gauge the robustness of a given solution because it is the accuracy rate that corresponds to complete randomness. Since there are 6 different ISUP scores, one could randomly categorize biopsy images and achieve an average 16.7% accuracy rate. The last model achieved a test accuracy rate of 63.3% on the test dataset, which is 3.8x greater than the accuracy that could be achieved by randomness.

2. Justification

The stated and accepted objective in the project proposal (see document "proposal.pdf" in the main repository) was to achieve a Kaggle submission score higher than the average public leaderboard score, which is 71.9 out of a possible 100. The final model generated a public Kaggle submission score of 79.0 (please find the public leaderboard here. Please find a link to the author's final notebook with public and private submission scores here). The author is very happy to have achieved the originally set objective and in the author's opinion, having done so demonstrates that the final model and solution is accurate enough to have adequately solved the problem.